НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Фізико-математичний факультет
(повна назва інституту/факультету)

Кафедра загальної фізики та фізики твердого тіла
(повна назва кафедри)

«На правах рукопису»
УДК 539.218/.219.2-022.513.2](043.3)

«До захисту допущено»
Завідувач кафедри

_____ _____
(підпис) (ініціали, прізвище)

“___”_____20__ р.

Магістерська дисертація

зі спеціальності (спеціалізації) 104 фізика та астрономія _____
(код і назва спеціальності)

на тему: Закономірності спікання металевих наночастинок в трьохвимірній Монте-Карло моделі _____

Виконав: студент 2 курсу магістерського рівня, групи _ОФ-71мн_
(шифр групи)

Скляров Євгеній Сергійович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник д. фіз. мат. н., проф. Горшков В. М._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент д. фіз. мат. н., проф. Решетняк С. О. _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

АНОТАЦІЯ

ДО МАГІСТЕРСЬКОЇ ДИСЕРТАЦІЇ

СТУДЕНТА ФМФ, 2 КУРСУ МАГІСТЕРСЬКОГО РІВНЯ ГР. ОФ-71мн

Склярова Євгенія Сергійовича

На тему: «Закономірності спікання металевих наночастинок в трьохвимірній Монте-Карло моделі»

Науковий керівник д. фіз. мат. н., проф. Горшков В. М.

Мініатюризація електронних пристроїв потребує вирішення багатьох технологічних проблем. Наприклад, охолодження нанопристроїв, виготовлення їх елементів на базі багатоатомних молекул і навіть з'єднання цих елементів у цілісну електронну систему нанорозмірних масштабів потребують нестандартних підходів на основі глибоких фундаментальних досліджень. Одній з задач такого типу, а саме проблемі спікання наночастинок благородних металів, присвячена ця магістерська дисертація. Особливість досліджених систем наночастинок в тому, що вони не знаходяться в прямому контакті між собою. Така ситуація реалізується у випадку, коли вони заповнюють полімер, яким прокладаються провідні доріжки в наносхемах. Подальше прогрівання цих доріжок призводить до випаровування полімеру і встановлення контактів між нанокластерами. Якість кінцевого стану утворених доріжок (провідність та механічна стійкість) залежить від режиму прогрівання: часу прогрівання та охолодження, максимальної температури нагріву, розподілу нанокластерів за розміром в початковому стані.

В роботах попередників задачі спікання в описаній постановці досліджені для шару наночастинок. В таких двовимірних моделях неможливо дослідити та оптимізувати еволюцію морфології реальної тривимірної системи нанокластерів, коли значно розширюється діапазон напрямків встановлення контактів з відповідними витратами значної маси на заповнення міжкластерного простору, і постає проблема запобігання значних каверн в провідній доріжці, які знижують їх механічну стійкість. Тому саме дослідженню спікання в тривимірних полімерних середовищах заповнених металевими наночастинками присвячена представлена магістерська дисертація.

SUMMARY

TO MASTER'S DISSERTATION

STUDENT FMF, 2 COURCE MASTERY GR. OF-71mn

Skliarov Yevhenii

Theme: «Patterns of sintering of metal nanoparticles in the three-dimensional Monte-Carlo model»

Scientific supervisor: d. phys. math. sci., prof. Gorshkof V. M.

In terms of the global tendency to electronic devices miniaturization various technological issues occur. For example, nanoparticles refrigeration, production of nanoparticles elements on the polyatomic molecules basis, and even the combination of these elements in a holistic nanoscale electronic system, require unconventional approaches based on fundamental research. This Master's Dissertation raises one of the key problems in this branch – the sintering of noble metal nanoparticles. The absence of the direct contact between nanoparticles is a hallmark of the systems we researched. It occurs when nanoparticles fill the polymer, which the leading paths in the nanoscale are laid by. The following warming-up of these paths leads to the polymer evaporation so the contacts between nanoclusters appear. The quality of the final state of the paths (conductivity and mechanical resistance) depends on the heating mode: the heating and cooling time, the maximum heating temperature, the distribution of nanoclusters by their size in the initial state.

In existing works of scholarship the sintering issues are investigated for nanoparticles layer. There is no possibility to investigate and optimize evolution of the real three-dimensional nanoclusters system morphology in this type of the surface models. In this case the range of directions for establishing contacts with the corresponding costs of a significant mass for filling the intercluster space is considerably expanded and the problem of preventing significant cavities in the conductive path that reduces their mechanical stability appears. That is the reason why this Master's Dissertation investigates a process of sintering in three-dimensional polymeric environments filled with metal nanoparticles.

ЗАВДАННЯ

на магістерську роботу студента

Склярова Євгенія Сергійовича

1. Тема роботи: Закономірності спікання металевих наночастинок в трьохвимірній Монте-Карло моделі

Керівник роботи: д. фіз. мат. н., проф. Горшков В. М.

Затверджено наказом по університету від «25» березня 2019р. №958-с

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи: використання розробленого програмного забезпечення дозволяє визначити якість спікання нанокластерів з кубічною гранецентрованою граткою що щільноупаковані в багатошарову структуру.

4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити):

- Ознайомитися з процесом спікання для гранецентрованої кубічної гратки.
- Ознайомитися з чисельною моделю руху вільних та зв'язаних атомів.
- Розробити комп'ютерну программу для розрахунку динаміки спікання системи, що складається з нанокластерів.
- Порахувати тестову задачу в вигляді конфігурації Вульфа, для перевірки фізичної моделі
- Порахувати задачу спікання тривимірної щільно упакованої структури з нанокластерів сферичної форми, що мають гранецентровану кубічну гратку.
- Побудувати графіки, які описують процеси спікання залежнно від обраного сценарію спікання.

5. **Перелік графічного матеріалу:** 30 малюнків

6. **Дата видачі завдання** 26.03.2019р.

Календарний план

Назва етапів виконання магістерської дисертації	**Строк виконання етапів роботи**	**Примітка**
Перегляд літературних джерел, складання огляду літератури	10.03.2019-25.03.2019	
Пошук і добір фактичних матеріалів, їх групування та систематизація	26.03.2019-31.03.2019	
Написання коду комп'ютерної програми для розрахунку розпаду вістря вольфрамових голок	01.04.2019-20.04.2019	
Запуск тестового експерименту задля перевірки створеної фізичної моделі	15.04.2019-19.04.2019	
Проведення основних експериментів зі спікання	20.04.2019-26.04.2019	
Обробка та візуалізація отриманих результатів	25.04.2019-03.05.2019	
Опис отриманих результатів	04.05.2019-14.05.2019	
Попередній захист	15.05.2019	
Корегування роботи відповідно до зауважень комісії	16.05.2019-19.05.2019	
Остаточний захист роботи	20.05.2019	

Магістр _____ Скляров Є. С.

(підпис) (ініціали, прізвище)

Керівник роботи _____ Горшков В. М.

(підпис) (ініціали, прізвище)

Реферат

Об’єкт дослідження: спікання нанокластерів з гранецентрованою кубічною граткою, що щільно упаковані в багатошаровій структурі.

Мета роботи: розробка програмного забезпечення для розрахунку динаміки спікання нанокластерів, отримання результатів розрахунку.

Методи дослідження:

теоретичні: узагальнення даних стосовно теми дослідження на основі науково-методичної літератури та офіційних освітньо-наукових джерел;

практичні: написання коду програми для реалізації процесів спікання нанокластерів з гранецентрованою кубічною граткою, що щільно упаковані в багатошаровій структурі.

Завдання дослідження:

- Ознайомитися з процесом спікання та його особливостями для ГЦК гратки.
- Ознайомитися з чисельною моделю руху вільних та зв'язаних атомів.
- Розробити комп'ютерну программу для розрахунку динаміки спікання нанокластерів з ГЦК граткою в багатошаровій структурі.
- Порахувати тестову задачу (набуття сферичного кластеру конфігурації Вульфа), щоб переконатися в достовірності створеної фізичної моделі.
- Порахувати динаміку спікання нанокластерів в щільноупакованій багатошаровій структурі для різних обраних режимів спікання.
- Побудувати графіки отриманих результатів.

Результати та їх новизна

Розроблена комп’ютерна програма котра описує модель спікання нанокластерів з ГЦК граткою в багатошаровій структурі. Отримана залежність якості утвореного матеріалу після спікання залежно від обраного сценарію прогріву.

Ступінь впровадження: робота є науковою базою для подальших досліджень.

Рекомендації щодо використання результатів роботи та область їх застосування: результати роботи можуть бути використані в мікро- та наноелектроніці, де використовується спікання наночастинок благородних металів.

Значення роботи та висновки: розроблена комп’ютерна модель спікання нанокластерів з ГЦК граткою для різних режимів прогріву.

Ключові слова: *спікання, ГЦК гратка, чисельна модель, нанокластери.*

В роботі наведено: використаної літератури - 43, сторінок: - 116, рисунків-58.

Abstract

Object of research: sintering of nanoclusters with face-centered cubic lattice, densely packed in a multilayered structure.

Purpose: development of software for calculating the dynamics of sintering of nanoclusters, obtaining results of calculation.

Research methods:

theoretical: generalization of data on the topic of research on the basis of scientific and methodological literature and official educational-scientific sources;

Practical: writing program code for the implementation of sintering processes of nanoclusters with face-centered cubic lattice, densely packed in a multilayer structure.

Objectives of the study:

• Get acquainted with the sintering process and its features for the fcc lattice.

• Look at the numerical model of the motion of free and bound atoms.

• Develop a computer program for calculating the dynamics of sintering of nanoclusters with FCC lattice in a multilayer structure.

• Calculate the test task (obtaining a spherical cluster of Wolf configurator) to verify the authenticity of the created physical model.

• Calculate the dynamics of sintering of nanoclusters with FCC lattice in tightly packed multilayer structures for different selected sintering modes.

• Build graphs of the results.

Results and their novelty

A computer program was developed that describes the model of sintering of nanoclusters with FCC lattice in a multilayered structure. The dependence of the quality of the formed material after sintering is obtained depending on the chosen heat recovery scenario.

Degree of implementation: work is a scientific basis for further research.

Recommendations on the use of the results of work and the scope of their application: the results of work can be used in micro- and nanoelectronics, where the sintering of noble metal nanoparticles is used.

Value of work and conclusions: a computer model of sintering of nanoclusters with a face-centered cubic lattice for different heating modes has been developed.

Keywords: sintering, FCC lattice, numerical model, nanoclusters.

The paper contains: used literature -43, pages: -116, drawings -58.

ЗМІСТ

ВСТУП

Мініатюризація електронних пристроїв потребує вирішення багатьох технологічних проблем. Наприклад, охолодження нанопристроїв, виготовлення їх елементів на базі багатоатомних молекул і навіть з'єднання цих елементів у цілісну електронну систему наноромірних масштабів потребують нестандартних підходів на основі глибоких фундаментальних досліджень. Одній з задач такого типу – проблемі спікання наночастинок благородних металів- присвячена ця магістерська дисертація. Особливість досліджених систем наночастинок в тому, що вони не знаходяться в прямому контакті між собою. Така ситуація реалізується у випадку, коли вони заповнюють полімер, яким прокладаються провідні доріжки в наносхемах. Подальше прогрівання цих доріжок призводить до випаровування полімеру і встановлення контактів між нанокластерами. Якість кінцевого стану утворених доріжок (провідність та механічна стійкість) залежить від режиму прогрівання: часу прогрівання та охолодження, максимальної температури нагріву, розподілу нанокластерів за розміром в початковому стану.

Багато теоретичних досліджень процесу спікання присвячено випадкам, коли нанокластери мають прямий контакт між сусідами. Але при використанні золотої полімерної «фарби» проміжки між наночастинками не вдається зменшити вужче 3-4 атомних шарів. При цьому відносна орієнтація нанокластерів визначає вірогідність встановлення контактів між сусідами, яка варіюється в широкому діапазоні. Задача оптимізації режиму спікання в тому, щоб забезпечити реалізацію всіх можливих міжчастинкових контактів без подальшого їх руйнування. Такі деструктивні процеси також можливі оскільки динаміка поверхні наночастинок внаслідок поверхневою дифузії атомів самоузгоджено пов'язана з просторовим розподілом міжчастинкової пари з вільних атомів багатьма фізичними механізмами.

В роботах попередників задачі спікання в описаній постановці досліджені для шару наночастинок. В таких двовимірних моделях неможливо дослідити та оптимізувати еволюцію морфології реальної тривимірної

системи нанокластерів, коли значно розширюється діапазон напрямків встановлення контактів з відповідними витратами значної маси на заповнення міжкластерного простору і постає проблема запобігання значних каверн в провідній доріжці, які знижують їх механічну стійкість. Тому якраз дослідженню спікання в тривимірних полімерних середовищах заповнених металевими наночастинками присвячена представлена магістерська дисертація.

РОЗДІЛ І. Кристалічні гратки металів

1.1. Аморфні і кристалічні тіла

У твердих тілах атоми можуть розміщуватися в просторі двома способами:

Безладне розташування атомів, коли вони не займають певного місця один щодо одного. Такі тіла називаються аморфними.

Аморфні речовини мають формальними ознаками твердих тіл, тобто вони здатні зберігати постійний обсяг і форму. Однак вони не мають певної температури плавлення або кристалізації.

Впорядковане розташування атомів, коли атоми займають в просторі цілком певні місця, Такі речовини називаються кристалічними.

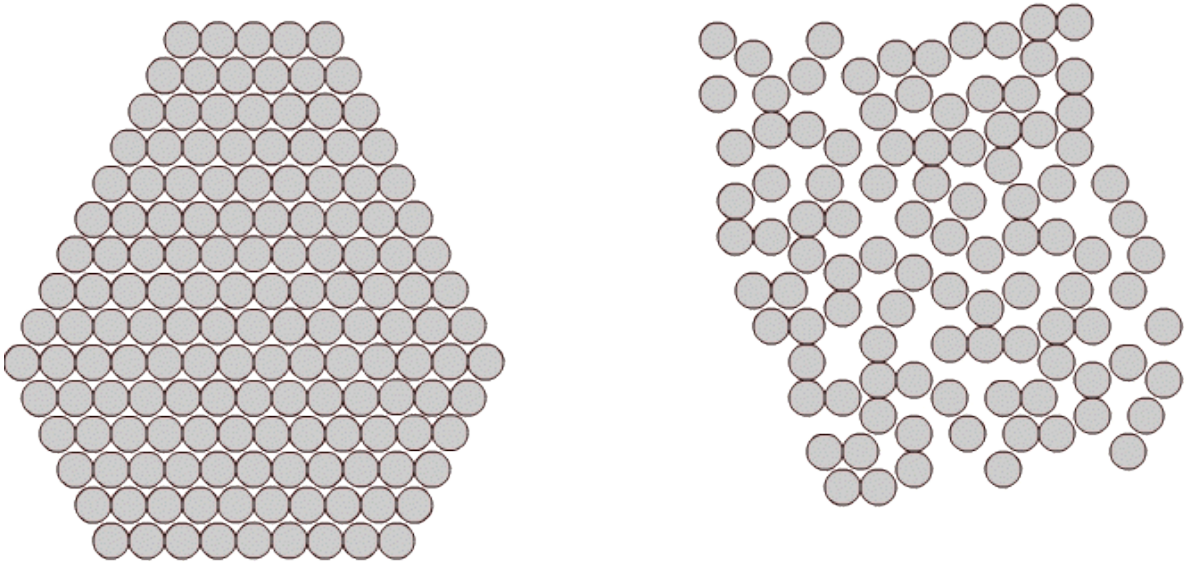
Атоми здійснюють щодо свого середнього положення коливання з частотою близько 10^{13} Гц. Амплітуда цих коливань пропорційна температурі.

Завдяки впорядкованого розташування атомів в просторі, їх центри можна з'єднати уявними прямими лініями. Сукупність таких пересічних ліній представляє просторову решітку, яку називають кристалічною решіткою.

Зовнішні електронні орбіти атомів стикаються, так що щільність упаковки атомів в кристалічній решітці досить велика.

Кристалічні тверді тіла складаються з кристалічних зерен - кристаллитов. У сусідніх зернах кристалічні решітки повернені відносно один одного на деякий кут [6-8].

У кристалітах дотримуються ближній і дальній порядки. Це означає наявність впорядкованого розташування і стабільності як оточуючих даний атом найближчих його сусідів (ближній порядок), так і атомів, що знаходяться від нього на значних відстанях аж до кордонів зерен (дальній порядок) [6].



а) б)

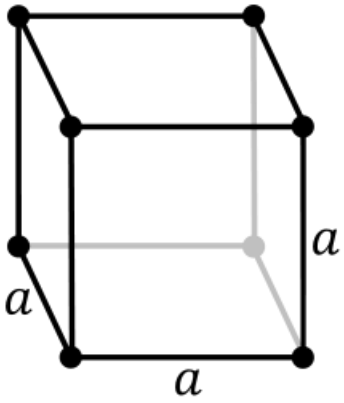
Рис. 1.1 Розташування атомів в кристалічній (а) і аморфній (б) речовині

Внаслідок дифузії окремі атоми можуть залишати свої місця в вузлах кристалічної решітки, однак при цьому впорядкованість кристалічної будови в цілому не порушується.

1.2. Основні типи кристалічних решіток

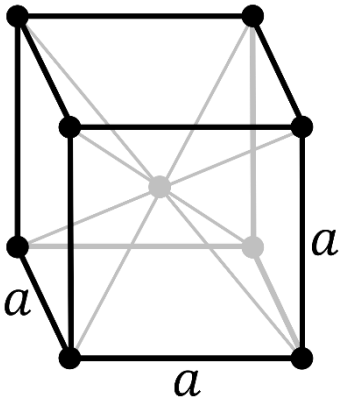
Всі метали є кристалічними тілами, що мають певний тип кристалічної решітки, що складається з малорухомих позитивно заряджених іонів, між якими рухаються вільні електрони (так званий електронний газ). Такий тип структури називається металевим зв'язком [7].

Тип решітки визначається формою елементарного геометричного тіла, багаторазове повторення якого по трьох просторових вісях утворює решітку даного кристалічного тіла.



а)

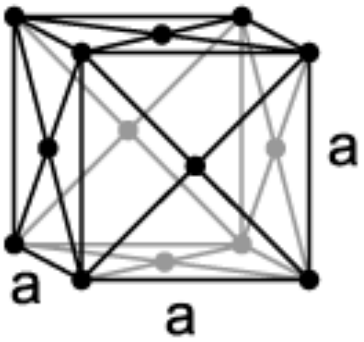
Кубічна (1 атом на комірку)



б)

Об'ємо-центрована кубічна (ОЦК)

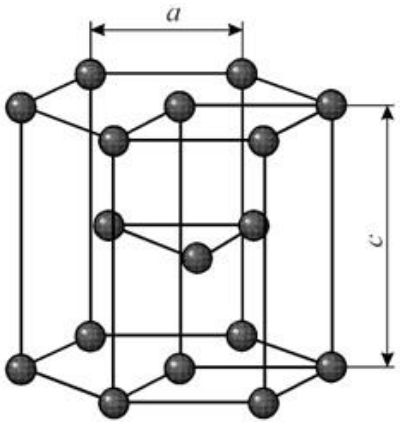
(2 атома на комірку)



в)

Гранецентрированна кубічна (ГЦК)

(4 атома на комірку)



г)

Гексагональна щільноупакована (ГЩ)

(6 атомів на комірку)

Рис. 1.2 Основні типи кристалічних граток металів

Метали мають відносно складні типи кубічних граток - об'ємно центрована (ОЦК), гранецентрированна (ГЦК) кубічні решітки.

Основу ОЦК-решітки становить елементарна кубічна осередок (рис. 1.2, б), в якій позитивно заряджені іони металу знаходяться в вершинах куба, і ще один атом в центрі його обсягу. На перетині його діагоналей. Такий тип решітки в певних діапазонах температур мають залізо, хром, ванадій, вольфрам, молібден і інші метали [8].

У ГЦК-решітки (рис. 1.2, в) елементарною клітинкою служить куб з центрованими гранями. Подібну грати мають залізо, алюміній, мідь, нікель, свинець і інші метали

Третім розповсюдженим різновидом щільноупакованих решіток є гексагональна щільноупакована (ГЩУ, рис. 1.2, г). ГЩУ-осередок складається з віддалених один від одного на параметр з паралельних зосереджених гексагональних підстав. Три іона (атома) знаходяться на середній площині між підставами [8].

У гексагональних решіток відношення параметра с / а завжди більше одиниці. Таку решітку мають магній, цинк, кадмій, берилий, титан і ін.

Компактність кристалічної решітки або ступінь заповнювання її обсягу атомами є важливою характеристикою. Вона визначається такими показниками як параметр решітки, число атомів в кожній елементарній комірці, координаційне число і щільність упаковки.

Параметр решітки - це відстань між атомами по ребру елементарного осередку. Параметри решітки вимірюється в нанометрів (1 нм = 10^{-9}м = 10 Å). Параметри кубічних грат характеризуються довжиною ребра куба і позначаються літерою а.

Для характеристики гексагональної решітки приймають два параметра - сторону шестикутника а й висоту призми с. Коли відношення с / а = 1,633, то атоми упаковані найбільш щільно, і решітка називається гексагональна щільноупакована (рис. 1.2 г). Деякі метали мають гексагональну решітку з менш щільною упаковкою атомів (с / а> 1,633) [8]. Наприклад, для цинку с / а = 1,86, для кадмію с / а = 1,88.

Параметри а кубічних грат металів знаходяться в межах від 0,286 до 0,607 нм. Для металів з гексагональної гратами а лежить в межах 0,228-0,398 нм, а з в межах 0,357- 0,652 нм.

Параметри кристалічних решіток металів можуть бути виміряні за допомогою рентгеноструктурного аналізу.

При підрахунку числа атомів в кожній елементарній комірці слід мати на увазі, що кожен атом входить одночасно в декілька осередків. Наприклад, для ГЦК-решітки, кожен атом, що знаходиться в вершині куба, належить 8 осередків, а атом, що центрує грань, двом. І лише атом, що знаходиться в центрі куба, повністю належить цій комірці.

Таким чином, ОЦК- і ГЦК-осередки містять відповідно 2 і 4 атома.

Під координаційним числом розуміється кількість найближчих сусідів даного атома.

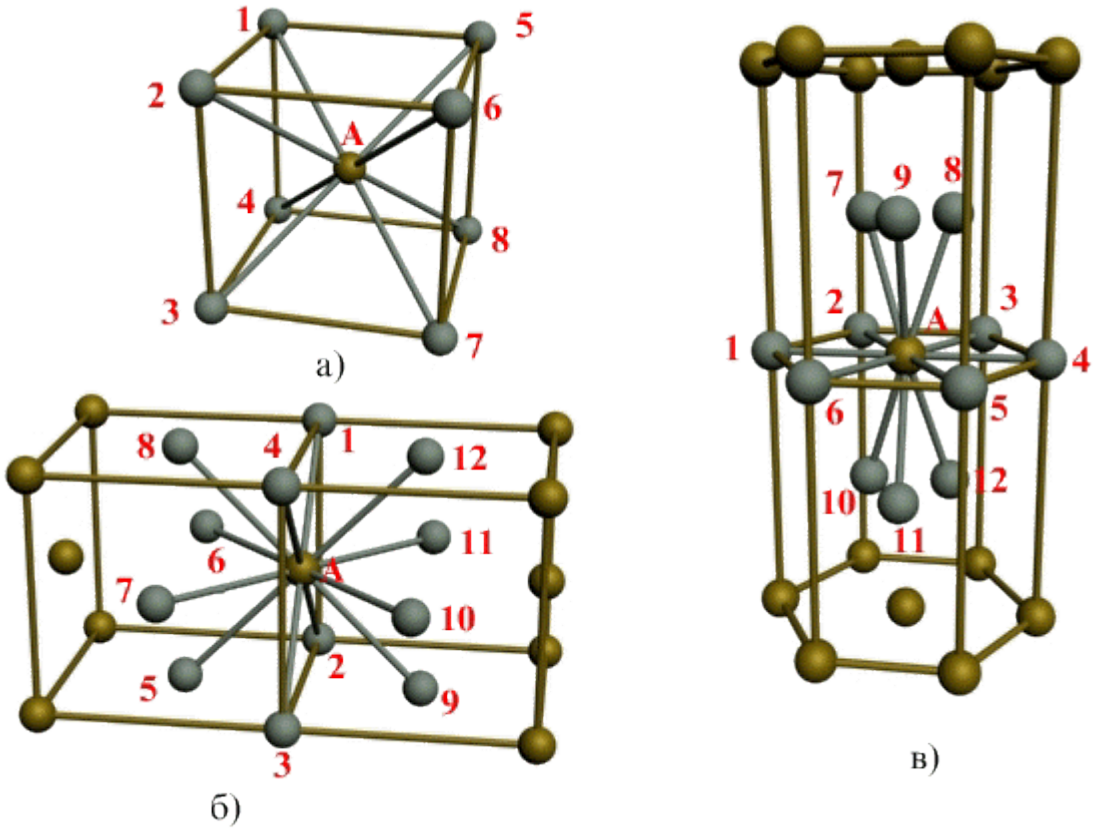


Рис. 1.3 Координаційне число в різних кристалічних решітках для атома А: а) - об'ємноцентрована кубічна (К8); б) – гранецентрована кубічна (К12); в) - гексагональна щільноупакована (Г12)

В ОЦК решітці (рис. 1.3, а) атом А (в центрі) знаходиться на найбільш близькому рівній відстані від восьми атомів, розташованих у вершинах куба, тобто координаційне число цієї решітки дорівнює 8 (К8).

У ГЦК решітці (рис. 1.3, б) атом А (на грані куба) знаходиться на найбільш близькому рівній відстані від чотирьох атомів 1, 2, 3, 4, розташованих у вершинах куба, від чотирьох атомів 5, 6, 7, 8, розташованих на гранях куба, і, крім того, від чотирьох атомів 9, 10, 11, 12, що належать розташованої поруч кристалічної осередку. Атоми 9, 10, 11, 12 симетричні атомам 5, 6, 7, 8. Таким чином, ГЦК решітки координаційне число дорівнює 12 (К12) [9].

У ГПУ решітці при с / а = 1,633 (рис. 1.3, в) атом А в центрі шестигранного підстави призми знаходиться на найбільш близькому рівній відстані від шести атомів /, 2, 3, 4, 5, 6, розміщених у вершинах шестикутника, і від трьох атомів 7, 8, 9, розташованих у середній площині призми. Крім того, атом А виявляється на такій же відстані ще від трьох атомів 10, 11, 12, що належать кристалічної осередку, що лежить нижче основи. Атоми 10, 11, 12 симетричні атомам 7, 8, 9 [9].

Отже, для ГПУ решітки координаційне число дорівнює 12 (Г12).

Щільність упаковки являє собою відношення сумарного обсягу, займаного власне атомами в кристалічній решітці, до її повного об'єму. Різні типи кристалічних решіток мають різну щільність упаковки атомів. У ГЦК решітці атоми займають 74% всього обсягу кристалічної решітки, а міжатомні проміжки («пори») 26%. В ОЦК решітці атоми займають 68% всього обсягу, а «пори» 32%. Компактність решітки залежить від особливостей електронної структури металів і характеру зв'язку між їх атомами [9].

Від типу кристалічної решітки сильно залежать властивості металу.

1.3. Кристалографічні напрямки і площини

Впорядкованість кристалічної будови в просторової решітці дозволяє виділити окремі кристалографічні напрямки і площини.

Кристалографічні напрямки - це характерні прямі лінії, що виходять з точки відліку, уздовж яких в кристалічній решітці розташовуються атоми. Точками відліку, можуть служити вершини куба, а кристалографічними напрямками - його ребра і діагоналі, а також діагоналі граней (рис. 1.4, а) [10].

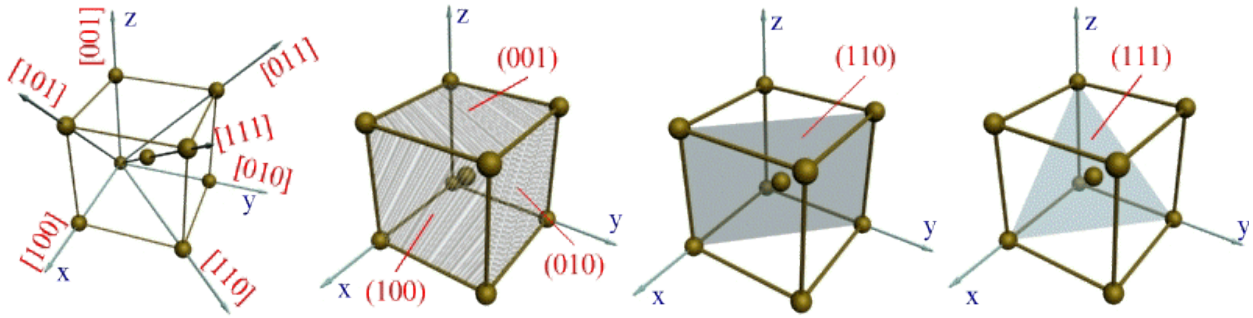


Рис. 1.4 Кристалографічні напрямки і площини в кристалічній решітці: а) - основні напрямки та їх позначення; б), в), г) - основні площини і їх позначення

Кристалографічними площинами є, наприклад, площини граней кубів (рис. 1.4, б), а також їх різні діагональні площині разом з розташованими на них атомами (рис. 1.4, в, г). Для ГПУ-решіток кристалографічними площинами можуть бути площині підстав (рис. 1.2, г).

Для визначення індексу будь-якого напрямку необхідно знайти індекс найближчого до цієї точки відліку атома, що знаходиться на даному напрямку. Наприклад, індекс найближчого атома уздовж осі ОХ позначається цифрами 100 (рис. 1.4, а). Ці цифри є координати згаданого атома щодо точки О, виражені через кількість параметрів уздовж осей ОХ, ОУ і ОZ відповідно [10].

Індекси напрямки OX і паралельних йому напрямків позначаються [100]. Відповідно напрямки OY і OZ позначаються [010] і [001]. Кристалографічні напрямків вздовж діагоналей граней XOZ, XOY і YOZ позначають [101], [110] і [011]. Користуючись зазначеної методикою, можна визначити індекс будь-якого напрямку. Наприклад, індекс напрямків вздовж діагоналі куба виразиться так: [111].

Для визначення індексу кристалографічної площині необхідно спочатку знайти координати найближчих точок її перетину з осями координат, проведеними з точки відліку О. Потім взяти зворотні їм величини і записати їх в круглих дужках у звичайній послідовності. Наприклад, координатами точок перетину з осями координат найближчої площині, паралельній площині XOY, вираженими через параметри решіток, є числа Ґ, Ґ, 1 (див. Рис. 1.4, б). Тому індекс цій площині можна записати у вигляді (001).

Індексами площин, паралельних площинах XOZ і YOZ, виявляться вираження (010) і (100) (рис. 1.4, б). Індекс вертикальної діагональної площині куба виразиться через (110), (рис. 1.4, в), а індекс похилій площині, що перетинає з усіма трьома осями координат на видаленні одного параметра, набуде вигляду (111) (див. рис. 1.4, г) [10].

РОЗДІЛ II. Спікання

2.1. Особливості ГЦК

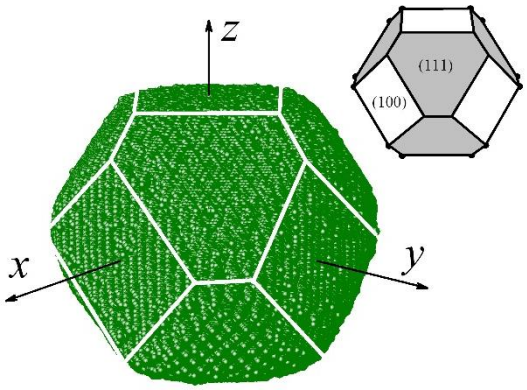


Рис. 2.1. Форма НЧ під час повільного дифузійного росту, отримана в численних експериментах [XX]. Розмір частинки вздовж вісей кординат рівний 63 постійним гратки, число атомів ~700 тисяч. На вставці конфигурація, побудована з площин типу (100) та (111) згідно з теоремою Вульфа.

Кожен атом в такій гратці має 12 найближчих сусідів, розташованих на сфері радіусом $a/\sqrt{2}$ (a - стала гратки). За умови наявності вакансії в найближчому оточенні центральний атом може перейти в вакантну позицію. Існує три види площин з мінімальною поверхневою енергією: це площини типу (100), (111) та (110). Проте конфігурація наночастинки, котра знаходиться в рівноважному стані з оточуючим її паром вільниї атомів(рис. 2.1), визначається лише лощинами (111) та (100). Цей результат має просту інтерпритацію. При енергії парної взаємодії атомів атомів в кристалічній гратці ε середня потенційна енергія атомів на гранях (111), (100) та (110) рівна -6ε. Проте, для динаміки форми нанокристалу, що росте важлива ще енергія зв'язку одиничних атомів, котрі осідають на ці площини.

а) б)

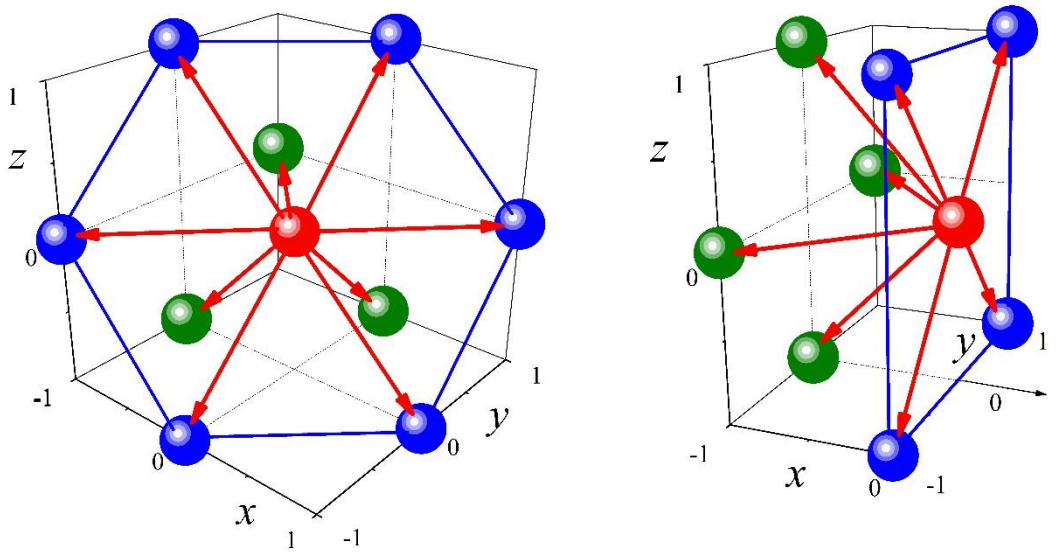


Рис. 2.2. Взаємодія поверхневих атомів (червоні стрілочки) з найближчими сусідами.

a- Заповнена грань (111): кожен з поверхневих атомів (червоних) взаємодіє з шістьмома атомами, котрі розташовані на цій грані(сині) та з трьтома атомами, що розташовані в об'ємі кристалу.; середня потенційна енергія зв'язку рівна -6ε.

b- грань (100): чотири взаємодії з поверхневими атомами та 4 з тими що в об'ємі (середня потенційна енергія зв'язку рівна -6ε).

Відповідно, при осадженні вільного атома на заповнену грань (111) енергія адсорбції рівна 3ε, для грані (100) - 4ε.

Атом, котрий осідає на площину(111) та починає формувати новий поверхневий шар, має енергію зв’язку 3ε: для нього всі атоми заповненого шару є «об'ємними», і число атомів, з якими він взаємодіє, дорівнює 3(рис. 2.2a). Аналогічно, для площини (100) отримуємо величину 4ε (рис. 2.2b). Енергія адсорбції атома на грань (100) рівна 5ε. В дифузійному режимі росту НЧ інтенсивніше приєднується до таких граней нових атомів(з малою

ймовірністю подальшого випаровування) призводить до росту клина обмеженого ділянками площин типу (111) та (100).

В подальшому будемо разглядати процеси спікання систем наночастинок, що мають початкову форму зображену на рис.1.

В реальних системах НЧ відрізняються за розмірами. Для пошуку оптисальних режимів спікання варто встановити механізми і характерний час утворення перемичок між сусідніми частинками, що залежать від взаємної орієнтації частинок в просторі та співвідношення їх розмірів. Окрім цього, вибір температурних режимів визначає стійкість утворених контактів: меньша за розміром частинка, котра «поєднує» більших за розмірами сусідів в єдиний кластер, може бути поглинена ними з порушенням цілісності кластеру.

2.2. Динаміка утворення перемичок

Розглянемо два випадки взаємної орієнтації частинок в просторі (рис. 2.3) різних розмірів.

а) б)

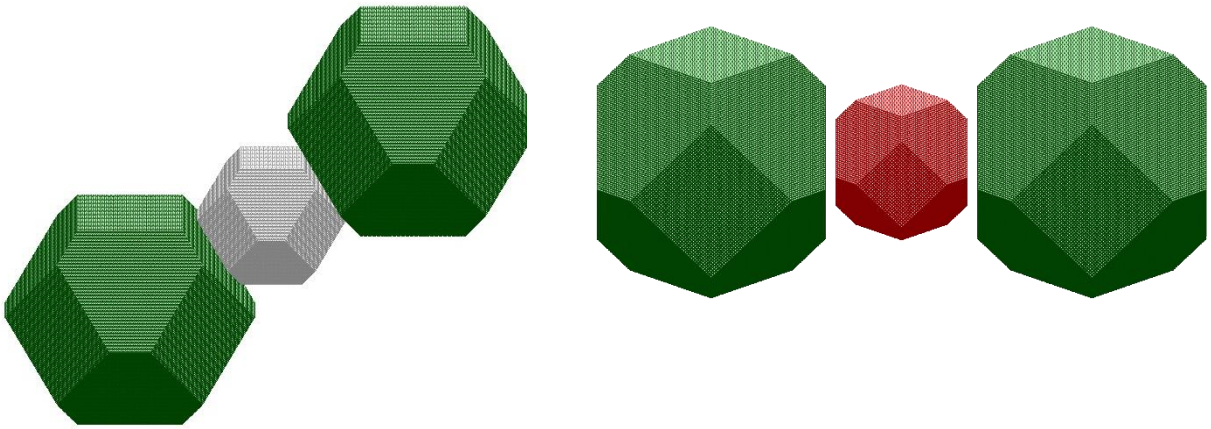


Рис. 2.3. Варіанти взаємної орієнтації НЧ, досліджуваних в роботі. Виникаюча в зазорі перемичка може містити від 6 до 12 шарів атомів.

Для відображення розглянутих конфігурацій як частин великої системи частинок, що спікаються було використано наступний прийом. Кожна окрема частинка заключена в кубі, обмеженому площинами(100). Ці площини віддалені від граней(100) частинки на 5-7 сталих гратки(залежно від величини R). Атоми, що випарувались з поверхні частинки, не можуть виходити за межі куб. Тобто, вільний простір, що оточує окрему НЧ на рис.2.3, обмежений в розмірах. Вільний простір всієї групи частинок являє собою об’єднанням вільних просторів їх елементів.

Проведемо порівняння властивостей граней (111) та (100), що знадобиться нам при інтерпретації результатів чисельних експериментів.

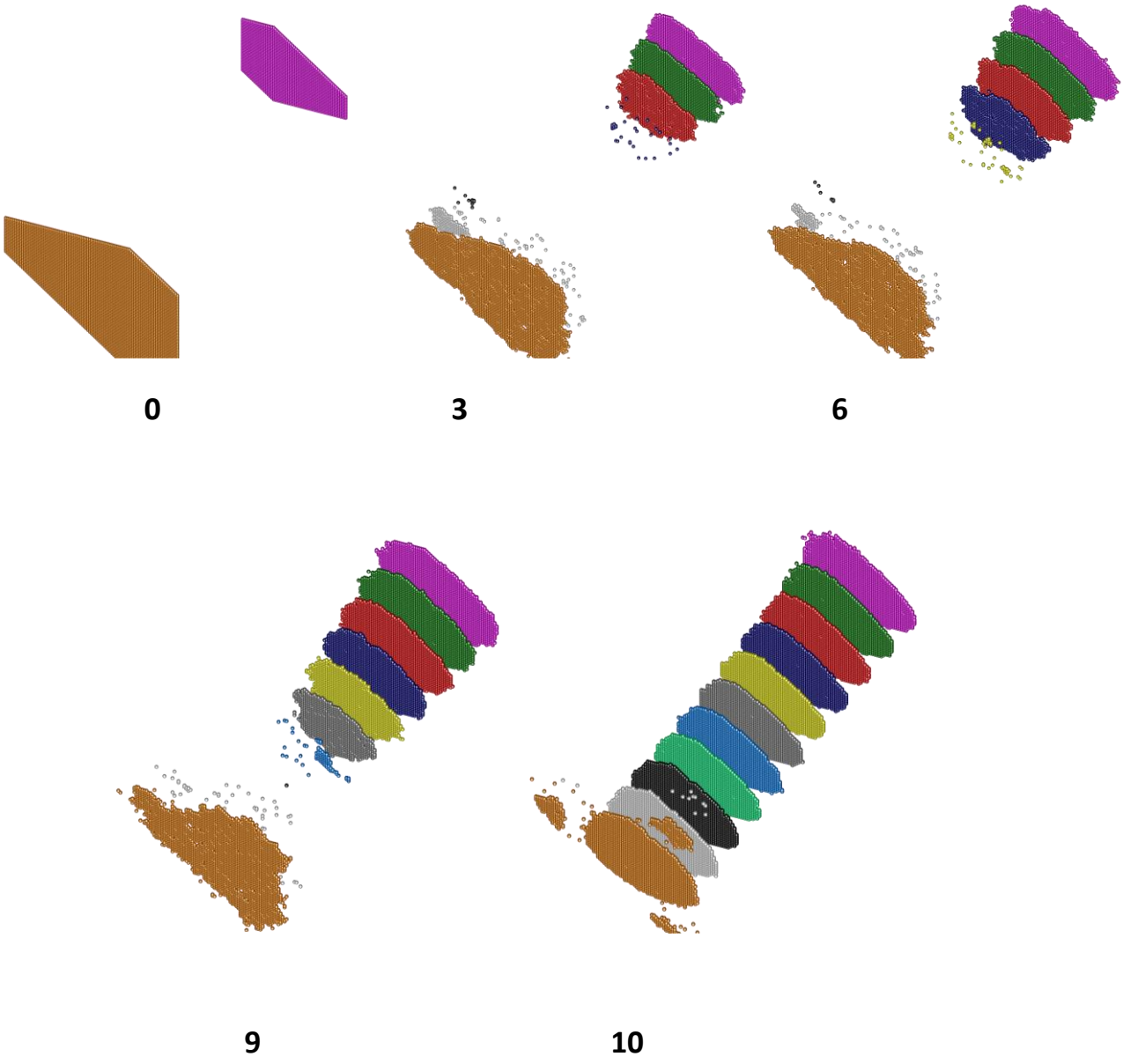


Рис. 2.4. Ріст перемичок поміж двома частинками , оберненими одна до одної гранями(111) – рис. 2.3а. Зазор між гранями $\Delta \sim 6a$, як результат між ними може бути сформовано 9 нових шарів. Радіус великих частинок $R = 50a$, малої $r_0 = 30a$. $\alpha = 0.8$. В початковий момент відображено грані (111) на малій та великій частинці, між якими буде утворено контакт. Час $t = 0$, $(3-6-9-10)\times 10^5$. Кожен утворений

- Вихід окремого атому з заповненої грані (111) потребує енергії 9ε , з грані (100) - 8ε . Під час нагрівання частинки концентрація пара з вільних атомів поблизу грані (100) вища, ніж у грані (111).
- Енергії адсорбції атома на заповнену грань рівні 3ε для (111) та 4ε для (100). Відповідно, коефіцієнт поверхневої дифузії атомів, адсорбованих

на (111) вище, ніж у атомів, адсорбованих на (100) (чим менша енергія зв'язку з підложкою, тим меньший середній час випадкового стрибку адсорбованого атому в сусідню поверхневу вакансію).

- Одношаровий кластер з адсорбованих атомів, утворених на грані(111), більш стійкий до розпаду, ніж такий самий кластер на грані (100): відрив одного атому з кластеру (при стрибку в сусідню вакансію) потребує енергії від ε до 5ε на грані (111) та від ε до 3ε на грані (100)-див. рис. 2.2.
- Відстань між шарами (111) рівна $a/\sqrt{3}$, між слоями (100) - $a/2$. Відповідно, на один атом в площині (111) доводиться площадка в $a^2\sqrt{3}/4$ та $a^2/2$ в площині (100).

Квазірівноважні грановані форми НЧ утворюються при відносно низьких температурах (параметр $\alpha = 1$ при отриманні конфігурації рис. 2.2.1 [1]). Утворення перемичок між НЧ є істотно нерівноважним процесом та відбувається при більш високих температурах, коли щільність пару з вільних атомів в міжчастинковому просторі досить висока: $\alpha = 0.7 - 0.8$. Саме через це механізми та час утворення перемичок між парами НЧ істотно залежать від їх взаємної орієнтації, що пов'язано з відмінностями в фізичних властивостях граней (111) та (100), вказаних вище.

Два випадки взаємної орієнтації частинок, обраних для чисельних експериментів (рис. 2.3), демонструють різноманітність фізичних сценаріїв формування контактів.

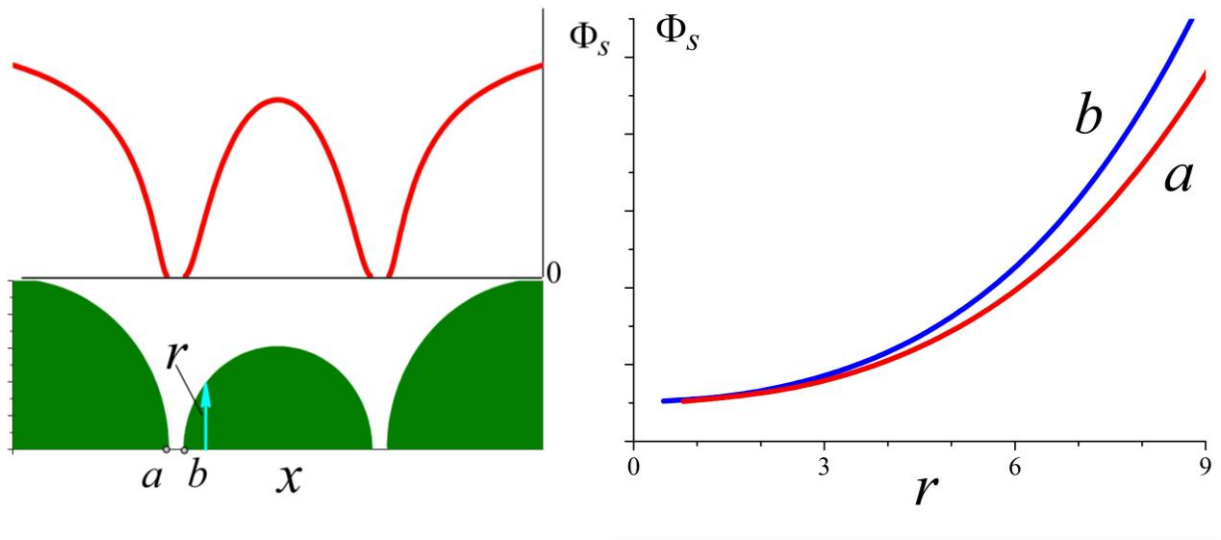


Рис. 2.5. Рішення електростатичної задачі для трійки НЧ. $R = 50a$, $r_0 = 30a$, ширина зазору $6a$. Зелений – ілюстрація геометрії системи. Червона лінія – щільність заряду (потоку $\Phi_s(x)$ в довільних одиницях. Права частина рисунку детально відображає залежність щільності потоку Φ_s поблизу полюсів частинок a та b в залежності від радіусу сферичного сегменту r.

А. Утворення перемички між НЧ, оберненими одне до одного гранями (111) (рис. 2.4) обумовлено прошарковим механізмом, детально розглянутим для випадку простої кубічної гратки в [1]. Він характерний тим, що в зазорі між частинками послідовно утворюються щільні додаткові шари атомів (рис. 2.4), звужуючи зазор до ~2-3 незаповнених шарів. Адсорбція вільних атомів на протилежні заповнені шари та їх подальші хаотичні зсуви вздовж шарів призводять як результат до утворення випадкового стабільного кластера-перемички, котрий швидко обростає новими атомами. Відмітимо, що при помітній різниці в розмірах наночастинок ноі шари утворюються переважно на меншій частинці. Обравши одну з випадкових реалізацій процесу, котра найбільш яскраво відображаєасиметрію в рості перемички. Проте в решті варіантів експериментів з обраними параметрами (рис. 2.4) вказана асиметрія стійко проявляється, хоч і в різній мірі, в 8 з 10 випадків. Час встановлення контакту T_c це випадкова величина, але вона змінюється в досить вузькому проміжку: $T_c \sim (0.9-1.2)\times 10^6$.

Асиметрія в утворенні перемичок безпосередньо пов'язана з механізмом їх утворення. Грані (111) обмінюються між собою атомами, що випаровуються з них та хаотично рухаються в зазорі. Проте за рахунок такого процесу неможливо вибудувати 4-5 додаткових шарів (рис. 2.4d). В обраній моделі система частинок оточена досить вузьким шаром вільного простору. При нагріванні НЧ випаровування атомів (особливо з ребер рівноважної форми - рис. 2.1) не призводить до значної втрати мас. Перенесення мас через вільний простір та збільшена з температурою поверхнева дифузія пов'язаних атомів заокруглюють частинки та можуть призводити до утворення нових щарів на гранях. Дійсно, такі шари утворюються. Але на гранях, не обернених до зазору, було зарегістровано не більше двох таких шарів.

Існує додатковий потік атомів в зазор з вільного простору, спричинений більш інтенсивним їх випаровуванням з суміжних граней (100), де енергія виходу з поверхні ніжче. Щільність додаткового дифузійного потоку атомів та прилягаючі до зазору ділянки поверхні наночастинок неоднорідна. Якісно проілюструємо це на простому прикладі (рис.2.6).

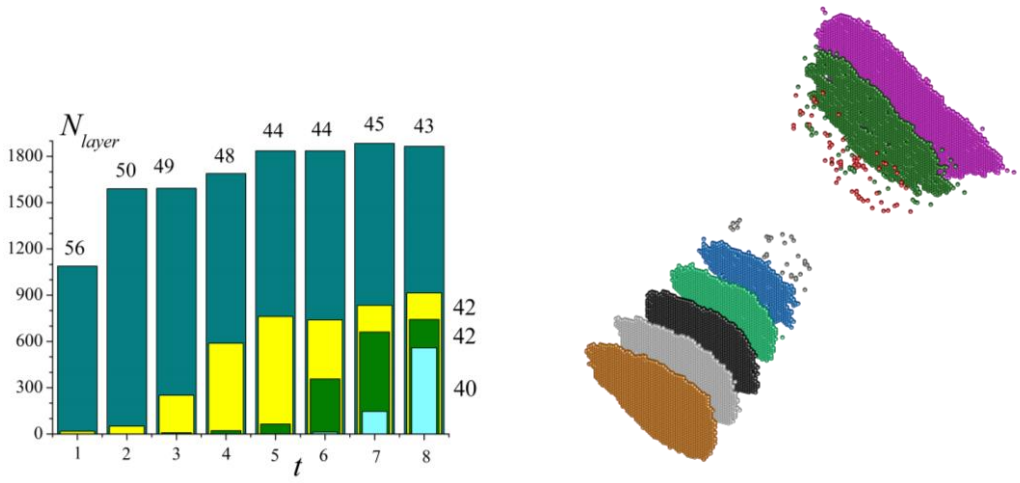


Рис. 2.6. Структура додаткових шарів, котрі виростають в зазорі між НЧ поблизу меньшої частинки (конфігурація рис. 2.3а). Зазор між гранями $\Delta \sim 6a$, радіус великих частинок $R = 50a$, малої $r_0 = 30a$. $\alpha = 0.8$. Кожен шар представлений власним кольором. Числа над глибоким блакитним та поблизу жовтого, оливкового та світло блакитного вказують відсотновий склад в утворених чотирьох додаткових шарах атомів, котрі в початковий момент входили в склад меньшої частинки. Час вказано в одиницях 10^5. До моменту $t = 8\times10^5$ поблизу більшої частинки утворено лише один додатковий шар з 60% вмістом «власних» атомів – права частина рисунку – (відображення в кольорі)

Відомо, що задача визначення щільності дифузійного потоку $\Phi_s(x, y, z)$ на поверхню частинки довільної конфігурації математично рівноцінна електростатичній задачі. Величина Φ_s прямопропорційна щільності поверхневого заряду заряда, розрахованого в припущенні, що поверхня частинки еквітопенційна. На рис. 2.5 видно, что в зоні «полюсів» a та b (зоні утворення перемички) $\Phi_{s,b}(r) > \Phi_{s,a}(r)$.

Перші додаткові шари мають формуватися переважно на меньшій частинці і утворювати на її поверхні нановиступи. Їх поява викличе перерозподіл дифузійноих потоків, сприяючи розширенню вже виникших шарів і утворенню нових на вершині виступу.

Рис. 2.6 відображає результати ще однієї випадкової реалізації утворення перемички в геометрії рис. 3а (початкові параметри такі ж, як і на рис. 2.4) і відображають дві важливі деталі в динаміці меньшої наночастинки.

- Значний вміст в знов утворених шарах «чужерідних» атомів, прибувших з великої сусідньої частинки, свідчить про інтенсивний обмін атомами через вільний простір.

- На рис. 2.6 (і 4) помітна виражена нестійкість в рості перемичок. Навелика початкова різниця в щільностях потоку вільних атомів на поверхню наночастинок в області їх полюсів a та b (рис. 2.5) наростає в часі в наслідок розвитку нановиступу на меньшій частинці. В електростатичній задачі(для змінившоїся геометрії системи) на виступі формується більш щільний поверхневий електричний заряд, що в нашому випадку відповідає збільшенню щільності потоку вільних атомів $\Phi_s(r)$ на поверхню виступу. Перерозподіл дифузійних потоків в зазорі на користь меньшої НЧ блокує утворення нових шарів на великій частинці(див. рис.4,6).

Все вищевказане відноситься лише до варіанту орієнтації частинок рис. 3а

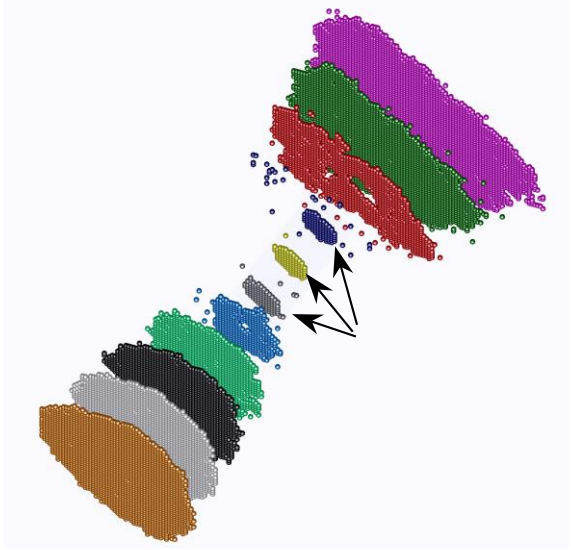


Рис. 2.7. Встановлення контакту малими кластерами(стрілки) на стадії завершення. $t=10^6$. Крайні нижній та верхній слої – початкові грані НЧ. Це результат ще однієї реалізації динаміки системи з параметрами рис. 2.4.

Зі збільшенням температури ($\alpha=0.7$ замість попередніх 0.8) час утворення перемички знижується до $T_c \sim (4-5)\times 10^5$. Величина T_c зменьшується також при звуженні зазору.

Наприклад, для розмірів частинок $R=35$, $r_0=21$, $\alpha=0.8$ та Δ відповідним 6 незаповненим шарам (111) (тобто всі розміри зменьшені в півтора рази відносно параметрів рис. 2.4, б) $T_c \sim (1.8-3.2)\times 10^5$. Варто звернути увагу на те, що при зменшенні зазору зростає відносне середньоквадратичне відхилення $\sigma^2 = \left[T_c - \langle T_c \rangle\right]^2 / \langle T_c \rangle^2$. Такий результат має просту інтерпритацію. Основний час для формування перемички витрачається на побудову протяжних в просторі додаткових шарів (рис. 2.4) до моменту, коли сусідні частинки розділяє ~3 не заповнених шари. Цей інтервал часу, T_{c1}, хоч і є випадковою величиною, проте дисить стабільний по тривалості в різних реалізаціях процесу. На заключній стадії в гру вступають кластери малих розмірів, котрі дрейфують по протилежним граням. В якийсь момент вони мають опинитись приблизно навпроти один-одного, щоб атоми з вільного простору встигли завершити формування стійкого контакту. Самец ця стадія, T_{c2}, значно варіюється по тривалості й визначає величинк σ^2 (при звуження зазору скорочується час T_{c1} без істотних змін статичних властивостей T_{c2}).

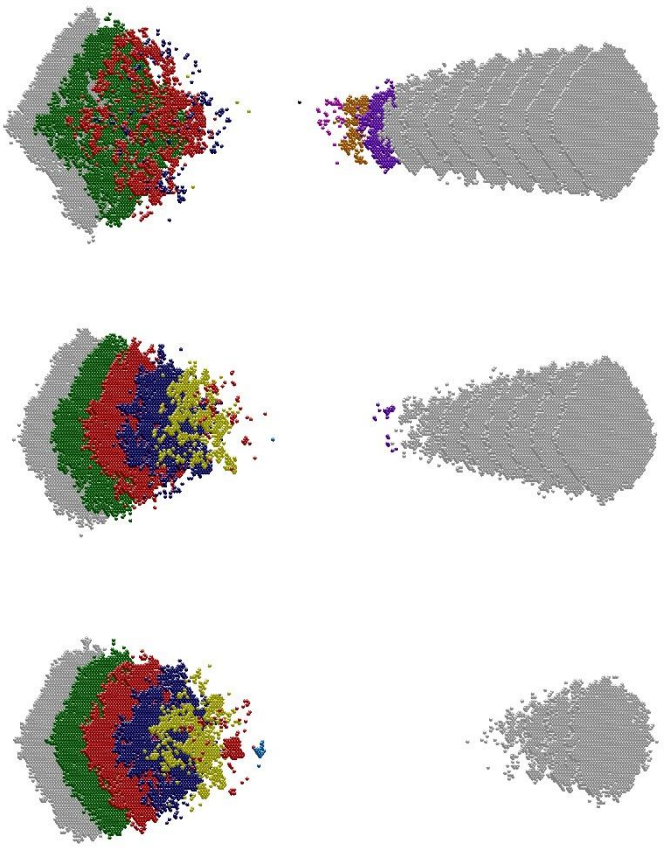


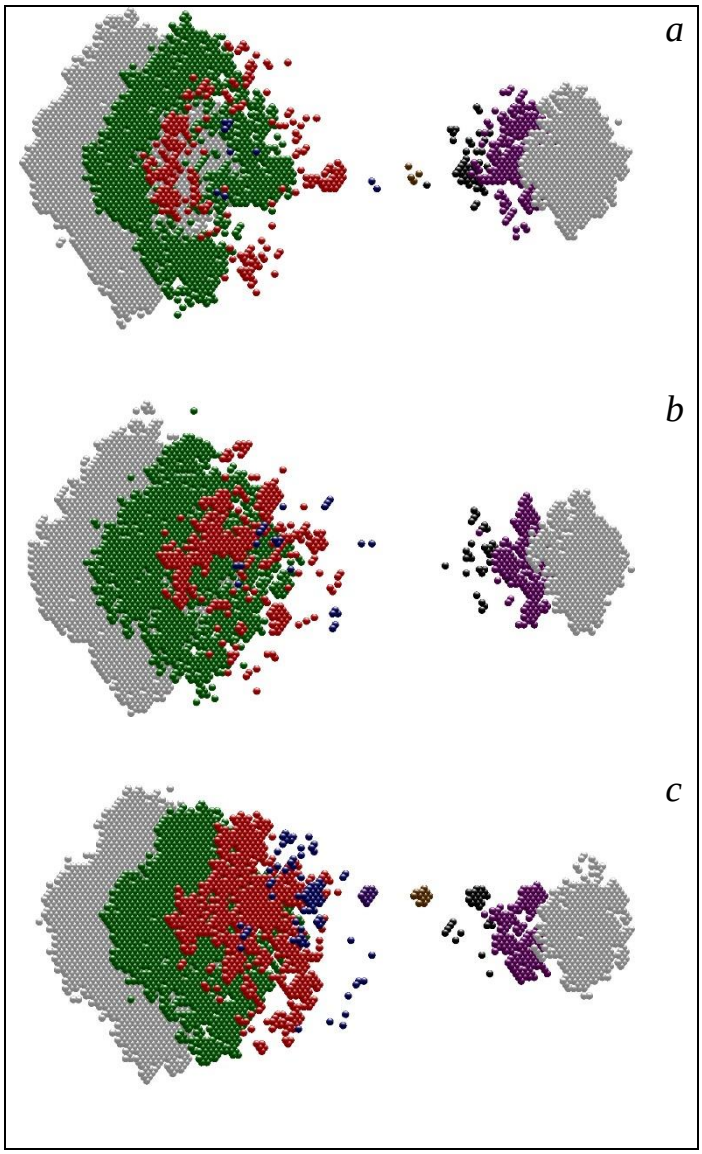
Рис. 2.8. Динаміка проміжку між частинками в конфигурації рис. 3b. $R = 50a$, $r_0 = 30a$. $\alpha = 0.7$. Ширина зазору $\Delta \sim 6a$,така ж, як і в варіантах рис. 2.4,2.6,2.7. З урахуванням меньшої відстані між шарами(100) зазор відповідає 11 незаповненим слоям. $t = 10^5$, 5×10^5, 1.5×10^6. Сірим зображені шари НЧ, заповнені в початковий момент, $t = 0$.

В. Обернемо частинки одна до одної гранями (111)-рис. 3b-з такою ж шириною зазору, як і на рис. 2.4,2.6,2.7: $\Delta = 6a$. Тепер замість 9 шарів (111) в зазор може ввійти 11 шарів(100). Щільність вакансій в зазорі не змінилась. Хоча грані (111) більше віддалені одна від одної, проте щільніше заповнені, ніж грані (100). Характерний результат чисельних експериментів зображений на рис. 2.8: перемичка між частинками не утворюється, а меньша частинка поглинається більшою. Провівши якісний аналіз динаміки системи в розглянутому варіанті.

Наночастинки обернені одна до одної поверхнями з меншою енергією випаровування (8ε). Додаткового притоку вільних атомів в зазор від суміжних граней, як це було в випадку рис. 3a, немає, адже у граней (111) енергія виходу вища за(9ε).

В початкові моменти часу проявляється механізм самокластеризації формування додаткових шарів, розглянутий в [23] для випадку простої кубічної гратки. Різниця між енергією випаровування та адсорбції для грані

(100) рівна 4ε (6ε для (111)). З цієї причини на гранях (100) більша ймовірність процесів, за яких атом з заповненого шару переміщується в вакансію сусіднього шару без втрати зв'язку з початковим шаром. Такі багатоступінчаті переходи призводять до утворення/поєднання кластерної структури, впровадженій в 2-3 слою зазору. (кластерний механізм є відносно швидким оскільки пов'язаний з стрибками атомів на малі дистанції). Деякий додатковий потік атомів в зону зазору пов'язаний з перенесенням маси через вільний простір і поверхневої дифузії від ребер кристалу(зон з малою поверхневою енергією).



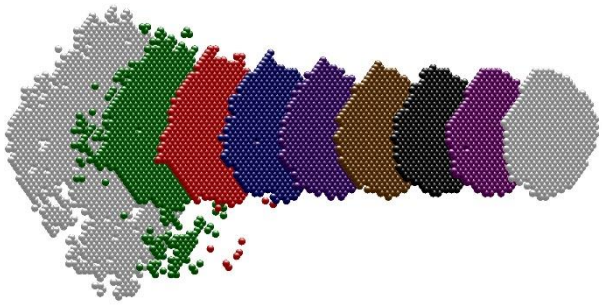


Рис. 2.9. Динаміка системи в конфігурації рис. 3b. $R=35$, $r_0=21$, $\alpha=0.8$ і $\Delta=4a$ відповідним 7 незаповненим шарам (100). $t=(30,60,97,120)\times10^3$. $t=97000$ - момент виникнення «голкового» контакту між частинками.

На цьому можливості заповнення зазору вичерпані. Лишившись ізольованою мала капля(з меньшою в середньому енергією випаровування атомів) поглинається більшою.

На основі вищесказаного можна очікувати, що перемичка утворюється при зазорі в 7 незаповнених шарів. По 2 слоя з кожного боку забезпечить швидкий механізм самокластеризації. Три шари, що лишилися в якийсь момент утворять контакт за допомогою випадкової «голкової» (рис. 2.7) структури з подальшим розвитком в часі.

Саме такий сценарій реалізовано на рис. 2.9. Для системи частинок з такими ж геометричними параметрами, але в орієнтації рис. 3а (в зазор Δ входить 6 незаповнених шарів (111)) час формування перемички T_c приблизно в три рази більший в наслідок повільного формування додаткових щільнозаповнених шарів.

Звуження зазору, як і в варіанті рис.3а, супроводжується зменьшенням часу T_c та ростом його відносного середньоквадратичного відхилення. Якщо в варіанті рис. 2.9 число не заповнених шарів в зазорі зменшити до 6, то $\langle T_c\rangle\approx4\times10^4$ і $\sigma\approx0.4$.

2.3. Динаміка спікання в шарі наночастинок

Розглянувши механізми утворення перемичок між парами НЧ. Цей процес є попередньою стадією спікання в системі наночастинок. При виборі оптимального температурного режиму слід враховувати, що частки розподілені за розмірами, за величиною зазору між сусідами і по взаємній орієнтації. Всі перераховані фактори впливають на час утворення перемичок T_c. Цей важливий параметр змінюється в широких межах, оскільки є випадковою величиною для кожної пари наночастинок навіть при всіх фіксованих початкових умовах. Крім того, статистичні параметри T_c залежать від морфології групи НЧ, що оточують виділену пару. Якщо розглядати шар наночасток, то в ідеальному варіанті спікання кожна з наночастинок встановлює контакти з найближчими сусідами, за якими в результаті поверхневої дифузії відбувається транспорт пов'язаних атомів, вирівнюючи розподіл маси по шару. Відзначимо, що в утворенні перемички істотну роль грає транспорт атомів, котрі випарувалися через вільний простір. Але після встановлення першого стійкого контакту динаміка системи значною мірою визначається поверхневою дифузією атомів (див. Рис. 4e, Рис. 9d -число атомів в зоні контакту різко збільшується за короткий час).

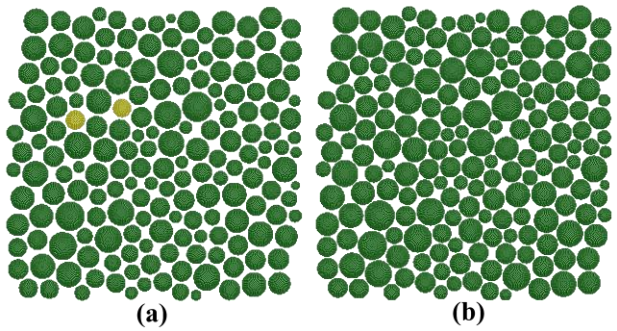


Рис. 2.10. Два типи випадкового заповнення розрахункової області розміром $150a \times 150a$. a – тип I, b – II. В обох випадках центри наночастинок мають одні й ті ж кординати, проте для II типу загальне число атомів приблизно на 15% вище, ніж для типу I. Мінімальна відстань між наночастинками рівно $2a$ (тип I) і a (тип II). Динаміка двох відмічених жовтих наночастинок, див. рис. 2.10a, для різних температурних режимів зображено нижче.

Значний розкид утворення можливих перемичок по часу T_c може призводити до того, що частка, яка встановила контакт з однією з частинок найближчого оточення, буде з часом настільки трансформована за формою, що контакти з іншими найближчими сусідами стануть неможливими.

Присутність в системі частинок менших розмірів ставить завдання їх «виживання» в процесі спікання. Контакти, встановлені між меншою часткою і більшими сусідами, можуть бути каналами транспорту атомів як до цієї частинки, так і від неї. Все залежить від співвідношення розмірів наночастинок і температури [24]. Необхідний вибір такої зміни температури за часом, $T(t)$, щоб не допустити масового поглинання малорозмірних наночастинок.

Фізичні принципи формування оптимального температурного режиму, тобто залежності $T(t)$, досліджую на прикладі моношару, який містить близько 200 наночастинок. Можливості численних експериментів обмежені загальним числом атомів в системі. Саме тому в моделі середній діаметр НЧ рівний $D_0 \approx 10a$. Щільність функції розподілу частинок за розмірами $p(D) \sim exp\left(-\frac{(D-D)^2}{2\sigma^2}\right)$, $\sigma = 2a$, тобто $\langle \Delta D^2 \rangle / \langle D \rangle^2 \approx 0.04$. Центри частинок лежать в одній площині ($z = 0$). Вільний простір для дрейфу випарувавшихся в процесі спікання атомів обмежено оболонкою $z = \pm z_s(x, y)$, віддаленою від поверхні частинок на $6a$.

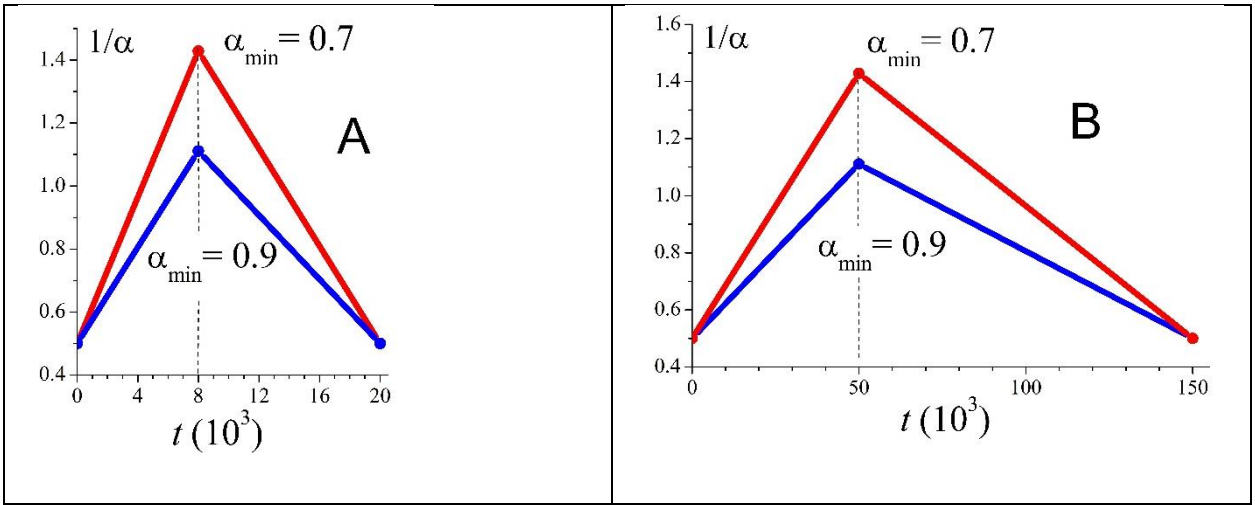


Рис. 2.11. Температурні режими що використовувалися в численних експериментах. А- короткий тепловой імпульс, В – тривалий в часі. В подальшому варіант розрахунку короткого імпульсу з $\alpha_{min} = 0.7$ бедемо позначати А07 й аналогічно в інших випадках.

Розглянуто два типи заповнення розрахункової області в момент $t = 0$ (Рис. 2.10). Тип I - зазори між частинками $\Delta \approx 2a$, і тип II - $\Delta \approx a$. Для кожного обраного теплового режиму $T(t)$ було проведено по 10 чисельних експериментів з незалежними випадковими реалізаціями розподілів частинок за розмірами і за координатами їх центрів.

Провівши чисельні експерименти для безлічі температурних режимів. Ілюстрацію основних закономірностей спікання системи наночастинок представимо для залежностей $T(t)$, що складаються з двох лінійних ділянок (див. Рис. 2.11-в нашій моделі параметр $\alpha \sim 1/T$). Нагрівання системи до максимальної температури супроводжується більш тривалим охолодженням. Початкові і кінцеві стани, коли $\alpha \sim 1.5 - 2$, відповідають дуже повільним поверхневим явищам, які практично не змінюють морфологію системи після в різній мірі результативного спікання.

Оптимальний режим спікання, відповідає тепловому режиму, коли будь-яка частка системи встановлює контакти з усіма найближчій сусідами. Такі

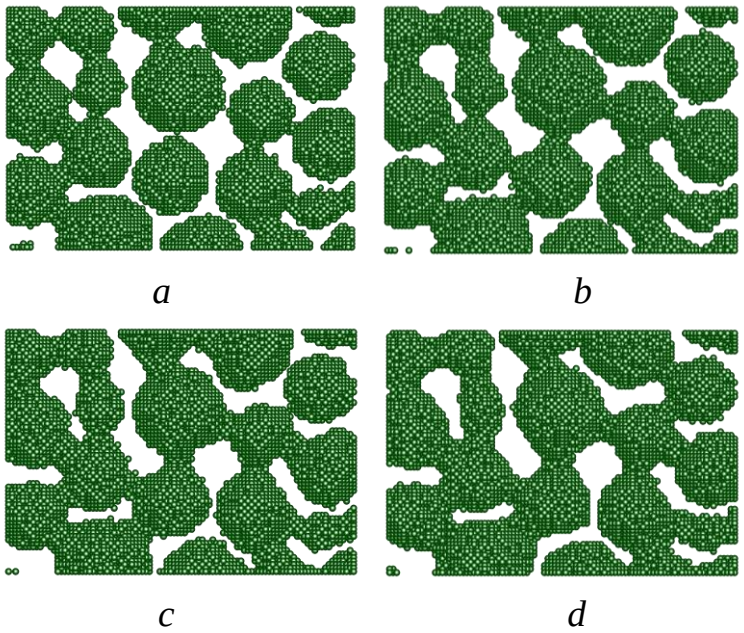
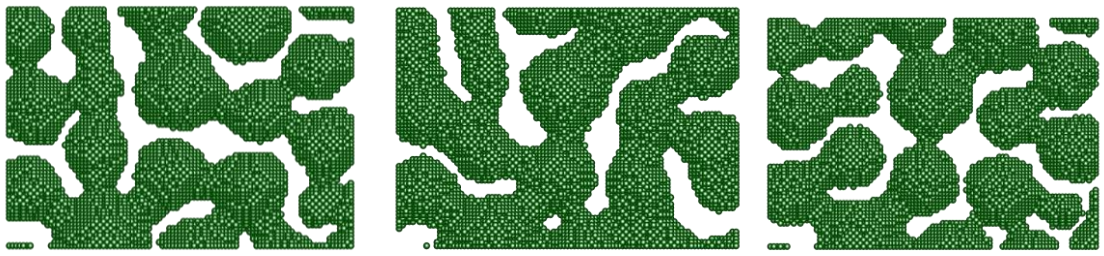


Рис. 2.12. Динамика контактов пары наночастиц, выделенных yellow в на чальном состоянии на рис. 2.10a. Тепловой режим A07, $t = (6; 6.5; 7; 7.5) \times 10^3$. Установленные на рис. 2.12d контакты сохраняются до завершения процесса спекания.

контакти не можуть бути встановлені одночасно, але бажано скоротити

часовий проміжок їх формування. В іншому випадку перенесення мас після встановлення перших контактів може повністю виключити утворення запізнілих в часі, але спочатку можливих контактів. Поставлену задачу вирішує інтенсивний короткий прогрів (див. Рис. 2.12 тепловий режим A07). На ньому показано еволюцію двох наночастинок, зазначених жовтим на початковому розподілі рис. 2.10*a*. Виділені частки встановлюють контакти з чотирма найближчими сусідами до моменту досягнення найбільшої температури.

Температурний режим A09 (відрізняється від A07 тільки значенням максимальної температури) недостатньо прогріває систему частинок; встановлені тільки два контакти лівою частинкою і три – правою (рис. 2.13a). При збільшенні тривалості нагрівання (зниження швидкості підвищення температури: режим B09- рис.13b і B07- рис. 2.13c) збільшуються тимчасові інтервали між моментами утворення можливих контактів. Затяжне за часом поверхневе перенесення мас до вже виникших контактів (до появи нових) змінює морфологію системи і ускладнює утворення цих нових контактів. Такий ефект особливо виражений при більшій максимальній температурі (поверхневої рухливості атомів): порівняйте рис. 2.13c з рис.12d.



a *b* *c*

Рис. 2.13. Контакти, утворені наночастинками, виділиними жовтим кольором на рис. 2.10*a* в різноманітних теплових режимах. *a*, *b*, *c*- температурний тежим A09 ($t = 12000$), B09 ($t = 50000$) , B07 ($t = 30000$). Встановлены контакти не руйнуються з плином часу, нових контактыв вказаны частинки не утворюють

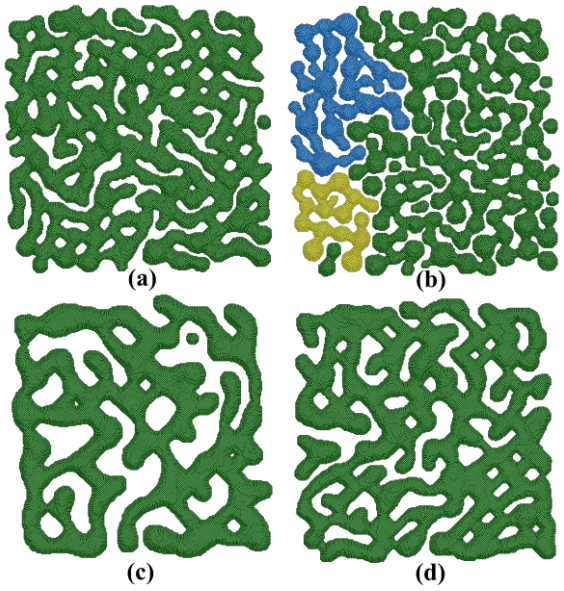


Рис. 2.15. Характерні структури з спікшихся наночастинок, отримані в різних температурних режимах для типу I початкового стану шару. a,b,c,d – температурні режими A07, A09, B07 ,B09

Мал. 2.14 демонструє прояв ефекту поглинання меншої НЧ більшими сусідами (небажаного для спікання). Зникнення такої НЧ свідомо формує дефект (порожнечу) в шарі спікання (рис. 2.14e). Межі отвору є зонами великої кривизни з малою енергією зв'язку атомів. В результаті поверхневої дифузії і відтоку атомів від кордонів такої області отвір буде розширюватися, якщо своєчасно не зменшити температуру для встановлення замороженого стану системи.

Рис. 2.14. Динаміка фрагменту системи при одному з випадкових реалізацій її заповнення за типом I. Температурный режим A07; $t = 0, 5700, 8000, 14000, 15000$.

Можна оцінити максимальний розкид наночастинок по розмірах для того, щоб виключити ефект поглинання менших за розмірами наночастинок.

Уявімо, що в початковому стані ми маємо дві великих НЧ з радіусом R і меншу між ними з радіусом r. При підвищенні температури мала НЧ утворила циліндричний контакт (див. Рис. 2.14d) с радіусом r_c. Якщо знехтувати початковими зазорами між наночастинками, то довжина контакту $l{\sim}2r$. Зі співвідношення $\frac{4\pi}{3}r^3 \approx 2r \times \pi r_c^2$ отримуємо, що $r_c \approx \sqrt{\frac{2}{3}}r$. Для збереження меншою НЧ необхідно, щоб транспорт атомів до зони примикання циліндра

до сфери (області, що має негативну кривизну) забезпечували великі частки. Імовірність стрибка пов'язаного атома уздовж поверхні (коефіцієнт поверхневої дифузії D_s), залежить від середнього числа сусідів (кривизни поверхні). Оцінкою критерію «виживання» малої частки може бути нерівність $D_{s,cilinder} < D_{s,sphere}$ що приводить до співвідношення для кривизни поверхонь $1/r_c$<2/R або $r \gtrsim 0.6R$. Ця груба оцінка задовільно узгоджується з результатами чисельних експериментів, виконаних для прямого ланцюжка з трьох частинок. З неї випливає рекомендація про граничну міру порушення монодисперсності наночастинок в шарі, $\delta^2 = \langle \Delta R^2 \rangle / \langle R \rangle^2$, для запобігання інтенсивного поглинання малоразмірної фракції шару: $\delta^2 \lesssim 0.04$, де R - радіус частинок. Розподілу часток за розмірами в наших численних експериментах як раз відповідають граничному показнику полідисперсності, $\delta^2 = 0.04$.

Після сказаного вище щодо властивостей локальних актів спікання природно виникають питання.

1. Чи слід прагнути до гранично високої монодисперсні для досягнення високої якості спікання моношару наночастинок в цілому?
2. Чи можна поліпшити якість спікання добавкою в систему наночастинок із середнім розміром $\langle D \rangle$ зовсім малих частинок, розмір яких $d \sim \left(\frac{1}{10} \div \frac{1}{7}\right) \langle D \rangle$?

На них відповім нижче. Порівнюючи результати спікання в цілому в залежності від температурних режимів (див. Рис. 2.15-показані кінцеві конфігурації системи після спікання, початковий розподіл наночастинок у всіх випадках один і той же).

Впадає в очі низька якість спікання при тривалому прогріванні шару (рис. 2.15c, d-режими нагрівання B07, B09). В силу фізичних причин, які ми обговорили вище, цей результат був очікуваний. Повільне підвищення температури призводить до того, що значна частина можливих контактів не встигає встановитися (хоча вся система вихідних НЧ спікається в однозв'язну область (єдиний кластер)). Поверхнева дифузія атомів від країв утворених отворів в плівці спікання призводить до їх значного розширенню на стадії охолодження.

Найбільш ефективний короткий теплової імпульс A07. Недостатній

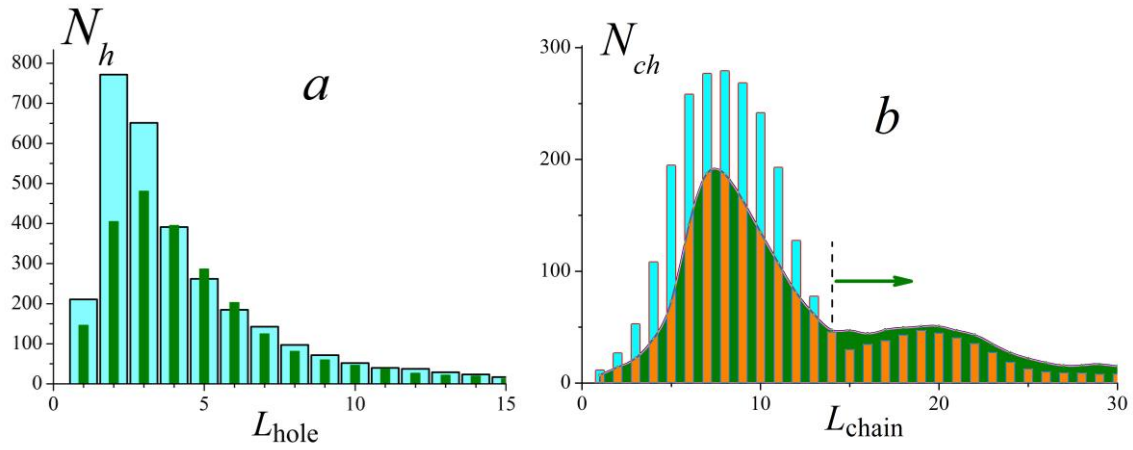


Рис. 2.16. Порівняння статистичних характеристик якості спікання теплових режимів A07, A09. *a* – розподіл по довжинам ланцюжків вакансій в площині $z = 0$, b – распределение по длинам цепочек из связанных атомов в этой же плоскости. Блакинтий – A09, оливковий – A07. Вертикальна пунктирна лінія виділяє область, в якій довгих заповнених ланцюжків в варіанті A07 більше, ніж в A09.

прогрів в варіанті A09 призвів до того, що система спікшихся частинок являє собою три однозв'язних кластера. Це результат випадковості процесу спікання, який проявився в одній з реалізацій[2]. Більш показовими усереднені по реалізаціям характеристики режимів A07 і A09 (рис. 2.16). Для цього в площині z = 0 системи визначений розподіл по довжинах ланцюжків з вакансій

(«порожніх» ланцюжків), $N(L_{hole})$, і ланцюжків з пов'язаних атомів, $N(L_{chain})$, які орієнтовані вздовж осей X і Y. Варіант A07 має кращі характеристики щодо варіанту A09 як за кількістю довгих порожніх ланцюжків (менше), так і за кількістю довгих заповнених ланцюжків (більше).

Розглянемо результати експериментів (рис. 2.17) для початкового заповнення типу II (див. Рис. 2.10b).

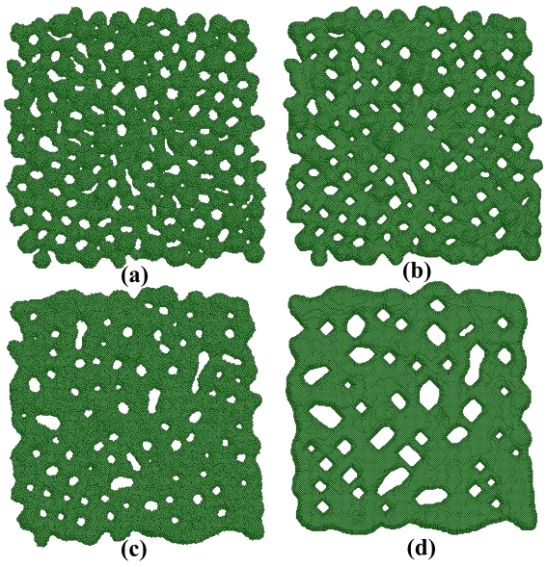


Рис. 2.17. Результати для початкового розподілу типу II. Верхня частина – характерна структура кластера в моменти часу t=8000 (максимум температуры) та t=20000 (завершення процесу) при температурному режиму спікания A08; нижня частина – режим B08, $t = 50000$ та 150000.

Зазори між частинками зведені до мінімуму. Здавалося б, що проблеми встановлення стійких контактів немає при різних температурних режимах. Однак навіть візуально видно, що тривалий прогрів / охолодження погіршує якість спікання (рис. 2.17c, d). Розподіл моментів встановлення контактів в широкому інтервалі по часу і тут супроводжується негативним ефектом утворення великих дірок. Звертаємо увагу на те, що ряд малих отворів, видимих на рис. 2.17c, з часом зникає- рис. 2.17d, а більші отвори розтягуються. Стінки отворів (протяжні уздовж осі Z) мають дві кривизни. Одна (завжди позитивна) - в площині, перпендикулярній площині z = 0. Друга - в самій площині z = 0. Ця кривизна як правило негативна. Для дірок малих розмірів домінує саме цей фактор, викликаючи транспорт пов'язаних атомів до зон негативної кривизни, що супроводжується звуженням отворів (або, принаймні, стабілізацією в часі їх розмірів).

Встановлення всіх можливих контактів за короткий проміжок часу (режим короткого прогріву) дробить весь вільний простір на дрібні осередки, які не

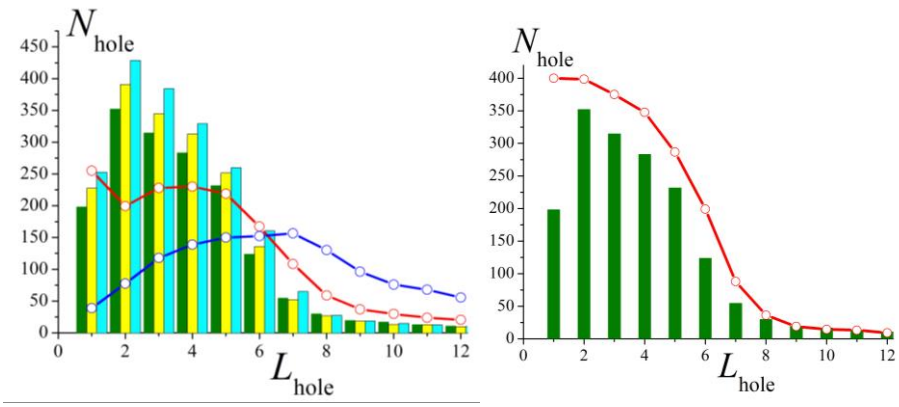


Рис. 2.18. Порівняння характеристик якості спікання шару частинок для початкового розподілу частинок типу II. a – Розподіл за довжиною пустих ланцюжків в кінці процесу спікання: оливковий –A07, жовтий – A08, блакитний– A09. Червоний – розподіл в момент досягнення максимуму температури в режимінагрівання B07 ($t = 5 \times 10^4$), синій – в момент закінчення процесу- $t = 2 \times 10^5$. b – Розподіл за довжиною пучтих ланцюжків при максимальній температурі нагрівання (червона лінія) в режимі A07 та в кінці процесу – оливкові стовпчики.

здатні розширюватись при подальшому охлажденіі- порівняйте рис. 2.17a і рис. 2.17b.

Мал. 18 являє кількісне порівняння характеристик якості спікання в різних температурних режимах.

Відмінності результатів для режимів A07, A08, A09 тільки в несуттєвою варіації числа коротких порожніх ланцюжків (рис. 2.18a). Червона лінія в області $L_{hole} > 6$ показує вади формування шару до моменту досягнення максимуму температури при розтягнутому в часі тепловому імпульсі B07. Відсутність достатнього дроблення вільного простору на дрібні осередки призводить на стадії охолодження до розширення дірок в шарі спікання (блакитна лінія - рис.2.18a).

Цей небажаний ефект пригнічений в режимі A07- див. рис. 2.18b. Великих дефектів значно менше в момент досягнення максимальної температури і вони стабільні в часі, тому що поверхнева дифузія не встигає розтягнути великі отвори за короткий час охолодження. Але його виявляється досить для

усунення дрібних дефектів (порівняйте оливкові стовпчики та червону лінію в фігурі 2.17b).

Дамо коротку відповідь на два поставлених вище питання. Чи слід прагнути до ідеальної монодисперсні частинок?

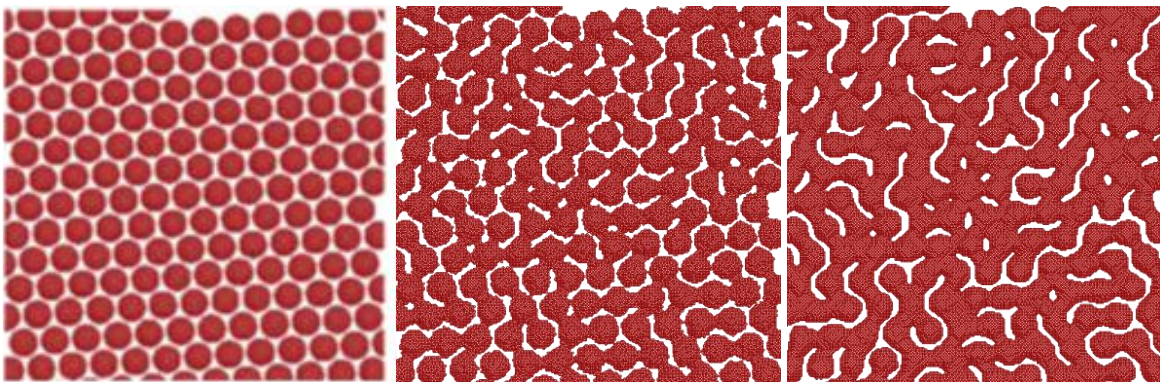


Рис. 2.19. Динаміка монодисперсної системи наночастинок в температурному режимі A08. a,b,c – розподіл наночастинок при $t = 0$, в момент досягнення максимуму

Рис.2.19 показує динаміку спікання для такого випадку. Радіус всіх частинок в початковому стані R = 5a, зазор між сусідніми частинками $\Delta \sim a - 1.5a$. Такий розподіл близький до типу II заповнення шару.

Порівняємо кінцеві стани полі- і монодисперсної систем для температурного режиму A08 (порівняйте рис. 2.17b і 2.19c). Результат спікання монодисперсної системи (рис. 2.19c) характерний великою кількістю протяжних змієподібних «розрізів». Деякі з них тягнуться від кордону розрахункової області до її середини. Різниця в статистичних характеристиках порожніх ланцюжків (рис. 2.20) дає кількісне уявлення про різницю в числі протяжних «розрізів» шару. Слід врахувати, що розподілу рис. 2.20 отримані для

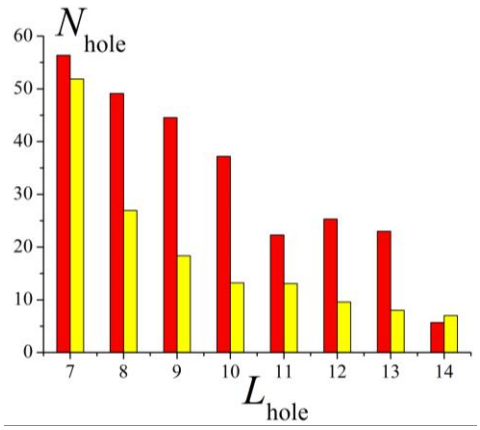


Рис. 2.20. Розподіл пустих ланцюжків для полі(жовтиний) та монодисперсної(червоний) систем наночастинок в кінці процесу спікання в температурному режимі A08.

прямолінійних порожніх ланцюжків (орієнтованих вертикально або горизонтально).

З урахуванням змієподібної форми розрізів шару реальна середня довжина порожніх ланцюжків на рис. 2.19c буде помітно вище величини, яка може бути отримана з даних рис. 2.20. Зазначена різниця в морфології кінцевих станів полі- і монодисперсної систем може проявитися на їх електричні властивості: провідність спочатку монодисперсної системи виявиться нижче відповідного значення для полідисперсної системи.

c *d*

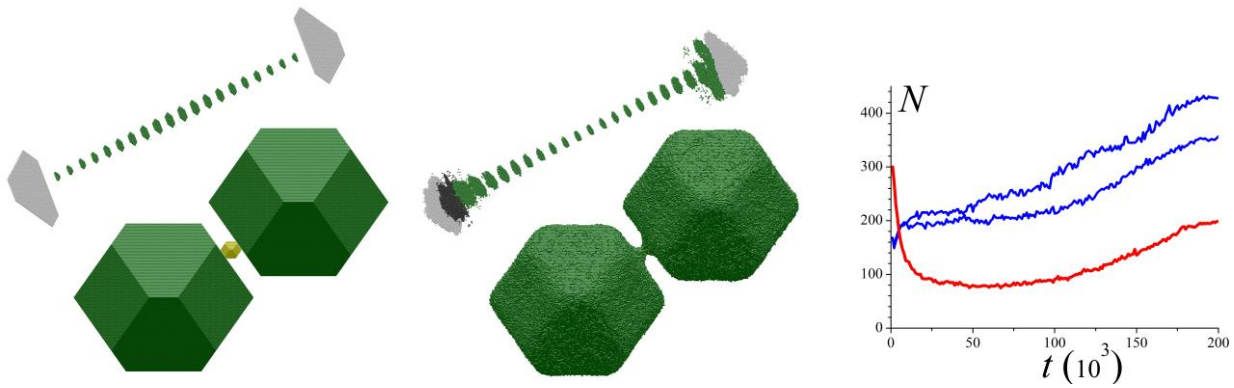
Рис. 2.21. Випадкові реалізації динаміки системи монодисперсних наночастинок з початковим розподілом рис.19a. Верхня частина (a,b)– режим B07, нижня (c,d) – B09. Структури шарів відображені в момент досягнення максимальної температури (a,c) і в кінці процесу (b,d).

Якісний аналіз даних, наведених на рис. 2.17-2.20 призводить до такого висновку. Час встановлення контактів $\langle T_c \rangle$ в монодисперсної системі (рис. 2.19a), створеної з відносно великих часток вище, ніж в полідисперсної (рис.10), що включає в себе фракцію малорозмірних частинок. Зростання $\langle T_c \rangle$ супроводжується збільшенням середньоквадратичного відхилення $\hat{\sigma} = \sqrt{\langle T_c - \langle T_c \rangle \rangle^2}$ (хоча відносне значення $\sigma = \frac{\hat{\sigma}}{\langle T_c \rangle}$ може зменшуватися-див. підрозділ 2,1). У цих умовах важче забезпечувати одночасність встановлення всіх можливих контактів. При цьому контакти, встановлені в початковій стадії спікання, спотворюють регулярність масопереносу в симетричній спочатку системі (рис. 2.19a) до такої міри, що

дефекти спікання не зникають з часом (порівняйте рис.19b і рис. 2.19c).

Більш тривалі режими прогріву системи (рис. 2.21) не покращують якості спікання щодо результату для полідисперсної системи (рис. 17). Механізми розширення дефектів спікання на стадії охолодження (особливо це помітно на рис. 2.21 a, b) ми обговорювали вище.

Таким чином, можна стверджувати, що для високої якості спікання слід використовувати полідисперсні склади наночастинок, але при виконанні критерію $\langle \Delta R^2 \rangle / \langle R \rangle^2 < 0.04$. Частинки меншого розміру можуть зіграти неоднозначну роль. Якщо така частка потрапить в тетраедричну (гексаедричну) порожнечу між великими частками в тривимірному випадку або в тригональну порожнечу в шарі наночастинок, то безсумнівна її позитивна роль. Але існує ймовірність, що вона виявиться в проміжку між великими частками, розсуваючи ці частинки на відстань, яка не допускає прямого встановлення контакту між великими частками. Тоді може реалізуватися сценарій рис. 2.14 з поглинання малої частки і утворенням дефекту спікання.



a *b* *c*

Рис. 22. Динаміка малої частинки з радіусом $r = 5a$, що розділяє великі частинки з $R = 50a$. Зазори між малою та великими частинками $\triangle \sim (1 - 1.5)a$. Параметр $\alpha = 0.7$. a – початковий стан, $t = 0$. Пряма, що поєднує центри частинок, орієнтована в напрямку (111). Вставка з верху– детальне відображення частинок з верху в шарах: сірий – слої наночастинок що, безпосередньо прилягають до зазору, зелений– слої малої частинки. b - $t = 2 \times 10^5$. c – залежності від часу числа частинок в шарах, що розділяють проміжок між двома великими частинками на 4 рівні частини. Червоний – центральний шар, блакитний – бокові шари.

Однак особливу роль може виконати фракція з гранично малим радіусом $r \sim \langle R \rangle / 10$. Як показали наші експерименти, такі частинки, навіть випадково впроваджені між двох великих (центри частинок на одній прямій), чи не поглинаються сусідками, а формують стійкий контакт, що знаходить просте фізичне пояснення.

Потік поверхневої дифузії атомів з боку великих часток встигає сформувати в проміжку між ними зону з великою негативною кривизною поверхні (великий енергії зв'язку-рис. 2.22b). Це призводить до зменшення коефіцієнта поверхневої дифузії в центральній частині системи і запобігання розриву перешийка внаслідок домінуючого на початковій стадії відтоку атомів від його центральної зони. Після досить тривалого періоду стабільності ($5 \times 10^4 \lesssim t \lesssim 10^5$, рис. 2.22c – червона лінія) починається період розширення перешийка.

Звернемо увагу на те, що розширення перешийка відбувається набагато повільніше, ніж його звуження на початковій стадії. Справа в тому, що в початкові моменти обидва головних радіусу кривизни поверхні в центральній зоні перешийка були позитивними. Малим енергіям зв'язку поверхневих атомів відповідає високий коефіцієнт дифузії і інтенсивний потік атомів від центру системи до зон примикання малої частки до великих (де в площині, що проходить через вісь системи радіус кривизни негативний). Формування негативної кривизни на всій поверхні перешийка ($t \gtrsim 10^5$) з одного боку запобігає його руйнування, а з іншого - обмежує приплив атомів з боку великих частинок внаслідок малої рухливості атомів на його поверхні.

Стабілізація перешийка внаслідок виникнення негативної кривизни вельми чутлива до розміру мінімальної частки. Ні в одному з 10 випадкових реалізацій чисельних експериментів перешийок не був зруйнований при r = 5a, але для r = 6a його розрив відбувався завжди. Середній час розриву $\sim 55 \times 10^3$.

РОЗДІЛ ІІІ. Моделювання

3.1. Метод Монте-Карло

Моделі на основі методу Монте-Карло є достатньо поширеним інструментом для побудови моделей в багатьох галузях фізики. Основна ідея полягає у тому, щоб створити серію станів системи з елементом випадковості, а не слідувати по прямій часу еволюції, що тільки збільшить фазовий простір системи. Послідовність конфігурацій має бути побудована в такий спосіб, щоб кожна конфігурація мала свою статистичну вагу. На практиці, такі моделі будуються за допомогою задання ймовірностей переходів між конфігураціями і послідуючим оновленням конфігурації.

Легко бачити, що розподіл Больцмана із попереднього розділу може бути утворений правильним вибором переходів, які задовольняють умові рівноваги системи. Після кожного переходу час системи інкрементується на τ, що є характерним часом системи.

Важливим аспектом у визначенні ймовірностей перетворень є визначення найшвидшого процесу в системі. Так, як метод Монте-Карло будується навколо головного циклу, який виконується до досягнення характерного результату (наприклад, утворення кластеру певної форми, проходження достатньої кількості часу). Кожен крок циклу зветься тіком. Найшвидший процес має ймовірність перетворення 1. Ймовірність всіх інших перетворень вимірюється відносно найшвидшого і обов’язково не перевищує 1. В противному випадку це свідчить про те, що найшвидший процес було обрано не вірно.

Основний цикл такої моделі зазвичай складається із наступних кроків:

1) Випадковим чином обирається перетворення і обраховується його ймовірність відносно найшвидшого процесу в системі.
2) Генерується випадкове число $N \in [0; 1]$ і порівнюється із ймовірністю перетворення. Якщо $N < P_i$, де P_i це ймовірність перетворення,

перетворення виконується і зміни зберігаються в моделі. В противному випадку не відбувається нічого.

3) Початок нової ітерації.

Цей метод організації обчислень є достатньо ефективним з точки зору програмування, оскільки оцінка доцільності кроку відбувається до його виконання, а система ймовірностей вірно балансує у швидкості різні процеси. Варто зазначити що кількість відкинутих кроків зазвичай сильно перевищує кількість виконаних, тому швидкодії оцінки доцільності (обрахуванню ймовірності та отриманню випадкового числа) варто приділити особливу увагу.

Випадку варіативності процесу перетворення, для нього можна побудувати граф ймовірностей, де корінь має ймовірність самого перетворення, а кожен наступний набір дочірніх нод зберігає інформацію тільки про співвідношення між ймовірностями перетворень на цьому рівні дерева. Таким чином для кожного складного перетворення утворюється простий і ефективний алгоритм визначення, що не потребує значних ресурсів і додаткового аналізу. Особливо ефективним в такому випадку є бінарне дерево, оскільки вибір гілки перетворення відбувається за час O(1) а не O(2n), де n – кількість варіантів перетворення на рівні, але треба пам’ятати, що вид дерева визначається алгоритмом перетворення.

3.2. Алгоритм Хошена-Копельмана

Алгоритм Хошена-Копельмана для визначення цілісності молекулярної структури є доволі простим для розуміння але не завжди тривіальним в реалізації.

Суть алгоритму зводиться до рекурсивного перебору усіх сусідів усіх частинок. Узявши довільну частинку системи, потрібно розглянути усі сусідні до неї частинки, потім усі, сусідні до її сусідів і так далі. Тоді, коли усіх сусідів буде перебрано, обирається наступна частинка із тих, що іще не були перебрані. Цей метод має дві проблеми: зациклення і масштабованість.

Виправити проблему із неможливістю вийти із рекурсії доволі просто: необхідно якимось чином помічати враховані частинки і більше не запускати аналіз для них. Проблема ж масштабованості потребує глибшого аналізу. Розглянемо систему N частинок. Якщо розмір кожного кластера є відносно малим, то навантаження на оперативну пам’ять комп’ютера буде незначним, але якщо вони являють собою один кластер, то глибина рекурсії може становити аж до самого N найгіршому випадку. Якщо оцінити N в реаліях необхідних експериментів, то: $N = 500000$, що дасть приблизно таку саму, в найгіршому випадку, глибину рекурсії, що фатально скажеться на програмі, а саме приведе до переповнення стеку виконання.

Для того, щоб уникнути вищевказаних проблем в конкретній реалізації було застосовано систему маркерів для частинок а також система черг для реалізації рекурсивного обходу в ширину, так як він технічно не потребує такого великого об’єму програмного стеку і показав себе значно швидшим за традиційний спосіб розв’язку рекурсивних задач для великої кількості частинок.

3.3. Конфігурація Вульфа

Нанокуби можуть мати характерні плазмонічні властивості, спричинені їхньою формою, та характерні каталітичні властивості, спричинені тим, що вплив {100} поверхонь максимальний. Отже, актуальним є питання протидії трансформації форми нанокубів, яке розглядається в останніх дослідженнях у сфері електронної мікроскопії та асоційованого аналізу. Було відмічено, що бар'єр до переміщення атома з ребра нанокуба до площини був ключовим у питанні контролю над трансформацією форми нанокуба. Більш комплексний нуклеаційний процес контролює перетікання форми в цілому. Завершені нанокуби зі {100} поверхнями мають нестійкі погано скоординовані ребра, кути і атоми на гранях. Отже, щоб зімітувати синтезовані, близькі до ідеальних нанокуби [3,36], ми починаємо з обрізаних срібних нанокубів, де всі атоми мають нейтронне число не менше 6. Це зображено на рис. 3.1 (a) для часового проміжку t=0.

Нижче зображена залежність температури та розміру перетікання у форму Вульфа. На рис. 3.1 (b) та 3.1 (c), ми відобразили «ширину» h100 (h111) між найбільш віддаленими від центру {100} поверхнями ({111} поверхні) на протилежних сторонах (кутах) нанокуба. Вони природнім чином перемасштабовані міжшаровим простором [a/ √2 для {100} поверхонь та шарів і a/ √(3/2) для {111} поверхонь та шарів] щоб відобразити зміну в кількості {100} та {111} шарів. Це визначення ідентифікує кожну грань з принаймні одним атомом в якості шару. Еволюція з обрізаного нанокуба до форми Вульфа включає формування нових {100} шарів на стороні початкового нанокуба та ерозію або відділення {111} поверхонь на кутах. Результат для більшого розміру N = 1584 атомів показує чіткі стадії цих процесів формування та відділення. Крім фінальної рівноважної площини, існує слабка площина для h100 та h111, що змінюється двома шарами. Це узгоджується з утворенням одного шару на кожну {100} поверхню, та повне відділення і переміщення {111} поверхні з кожного кута. Варто відзначити, що

формування шару лише на одній стороні, або переміщення шару лише з одного кута, але не з іншого, вочевидь рідкісний тимчасовий стан.

Щоб оцінити залежність T еволюції, ми представляємо характерні часові проміжки τrelax = τ100 and τ111, відображаючи істотну еволюцію h100 та h111, відповідно, від їхніх початкових значень [наприклад, зміна двома шарами, як показано на рис.3.1(b) та 3.1(c)]. Від цих характерних часових проміжків, ми оцінюємо ефективні енергії Арреніуса, Eeff. Fortrelax = τ111, що характеризують відділення кутової поверхні {111} та переміщення її атомів до {100} поверхні, аналіз Арреніуса результатів KMC для τ111 Eeff ≈0.7eV. Вважаючи, що нанокластер починає як ідеальний обрізаний нанокуб, кутовий атом у {111} поверхні переміщується до {100} поверхні, через що може вважатися перекрученою ділянкою на щільному кроці ребра для {100} поверхні. Бар'єр для першого кроку Eact =0.525eV відносно низький. Однак, на другому кроці щоб досягнути остаточної ділянки поглинання на {100} поверхні, що є E =+ 2φ вище початкової енергії ділянки, атом мусить перебороти енергетичний бар'єр cTD100 понад енергією заключного стану. Таким чином, загальний бар'єр переміщення атомів є Eeff =c100TD +E =0.875e, порівнюваний з результатами симуляції. Зауважимо що для вищої T, фактор ентропії, пов'язаний з термічним збудженням є значущим, і не включений у аналіз. [5]

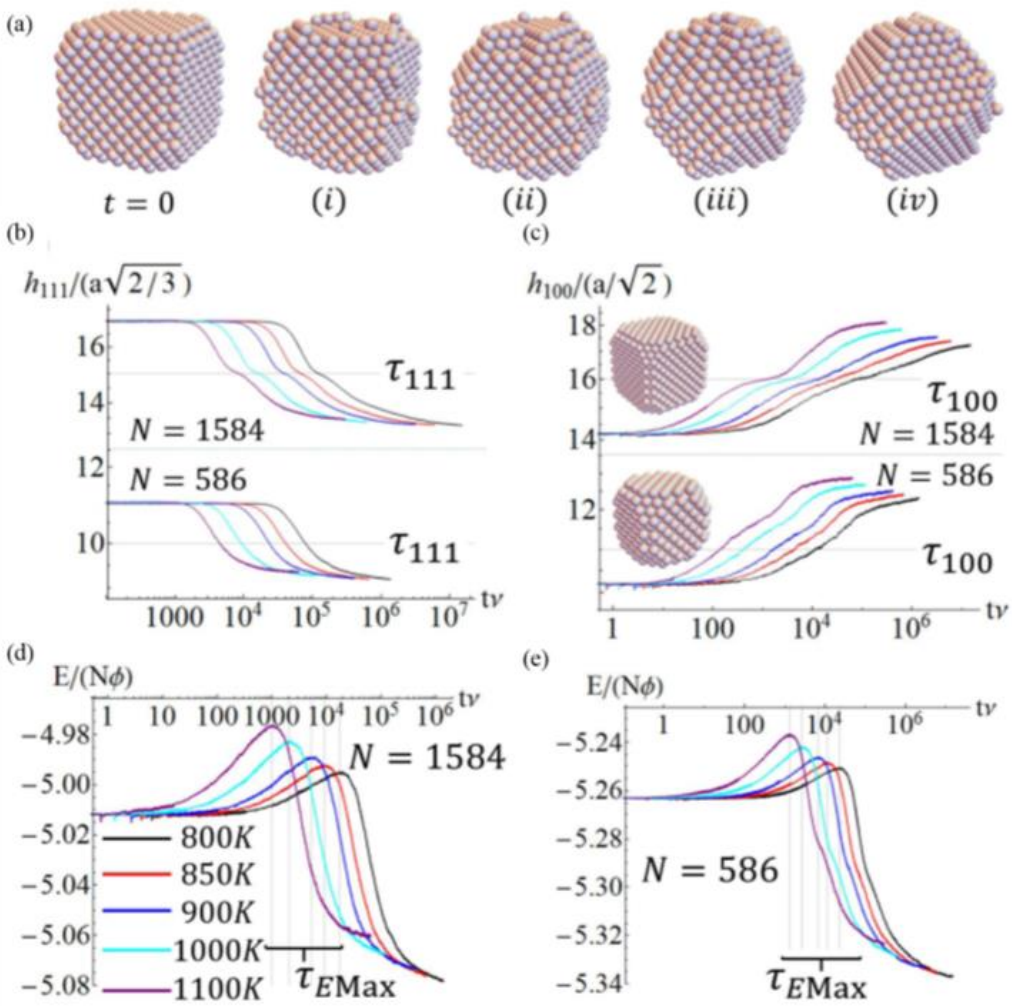


Рис 3.1 Переформування анокубу Ag: (a) конфігураційний знімок(N = 1584, T = 1100K). T – залежність часу еволюції масштабування $h_{111}(t)$ в (b), збільшення $h_{100}(t)$ на фігурі (c) і масштабування сумарної енергії на фігурі (e) та (d) для N=586,1584 в середньому за 400 випробувань.

Атоми звільнені з {111} поверхонь розпилюються на поверхні {100} і утворюють нові {100} шари, цей процес характеризується τrelax = τ100, для якого аналіз КМС результатів Арреніуса дає чіткий та високий Eeff ≈1.1eV. Тут природньо аналізується формування порівняно стабільного прямокутного тетрамера з атомів у {100} частинках. Врахуємо по-перше переміщення трьох з 12 атомів в початкову завершену {111} кутову поверхню до окремої {100} повехні, щоб сформувати тример. Це включає руйнування всіх восьми бокових ребер на {111} поверхні, але при цьому формування двох бокових ребер у

тримері, і підвищення координації для підтримки атомів на {100} поверхні для загальної зміни енергії E123 =+ 3φ. Дозволимо E4 позначити бар'єр для переміщення четвертого атома зі {111} поверхні до {100} поверхні, щоб стабілізувати тример. Цей процес контролюється останнім кроком для досягнення {100} поглинання ділянки, даючи бар'єр E4 =0.75eV. Це означає ефективний нуклеаційний бар'єр Eeff =E4 + E123 =1.42eV. [5] Однак, якщо тример на ребрі {100} поверхні, отже атом зі {111} поверхні може перестрибнути напряму всередину ділянки з двома боковими ребрами, що формують квадратний тетрамер, тоді E4 скорочується до 0.525eV і Eeff =1.20eV, цілком відповідно до результатів симуляції.

Також відслідковуємо загальну енергію нанокуба, E [Рис. 2 (d) та 2(e)], і визначаємо зміни часу τrelax = τEMax відповідно до пікової енергії. І пікова, і пізня стадії E більші для вищого T через ефект ентропії. Аналіз Арреніуса для τEMax: Eeff ≈0.72eV, що збігається зі значенням для τ111. Отож, енергетичний максимум відповідає ранній стадії руйнування {111} поверхонь, E знижується лише після того як нові {100} шари утворюються і ростуть.

Вищенаведені результати для еволюції h111, h100, та E отримані з широкої симуляції KMC, усередненої з кількох сотень спроб. Необхідно мінімізувати ефект значних коливань на нанорівні, і таким чином отримати точні результати для характерних часів та енергій Арреніуса. Зауважимо також, що майже бездоганна поведінка Арреніуса для характерних часів помічена понад випробуваних температур (див. Додатковий матеріал [27]). Ідентифікована поведінка Арреніуса дозволяє передбачення часових проміжків перетікання для нижчих T. Нуклеаційний процес з вищим Eeff ≈1.1eV буде з регульованою швидкістю, припускаючи що, наприклад, τrelax = τ100 ≈10−3.6, 10 −0.8, і 103.8 при 500, 400, і 300 K, відповідно, для N =1584, обираючи ν =1012.5 s−1. Ці оцінки є нижніми межами, оскільки Eeff має якимось чином підвищуватися для нижчого T. [5]

Приблизно оцінюючи розмір масштабів τrelax, грунтуючись лише на двох розмірах нанокуба: N =586 та 1584. Аналіз τrelax ∼Nβ відповідно до пізніх стадій процесу показує підвищення β для зниження T від β ≈1.3 за 1000 K до β ≈1.7 за 800 K. Відхилення для нижчих T є класичним значенням континууму β =4/3 і свідчить про присутність опосередкованого нуклеаційного процесу зі скінченним ефективним бар'єром [19,21]. Це спостереження відповідає тому, що еволюція h100, і особливо що енергія Арреніуса, виділена з τ100, контролюється утворенням нових {100} шарів.

3.4. Експеримент №1: відтворення конфігурації Вульфа

Одним з ключових критеріїв працездатності створеної моделі - здатність відтворити поведінку речовини без зовнішнього впливу. Якщо результат моделювання не повторить поведінку реальної системи, то модель побудована невірно. Для даного випадку було обрано об’єм кубічної форми із ребром у 100 сталих кристалічної гратки. В середині об’єму розташовано кластер кулеподібної форми. При значенні температурного коефіцієнту α=2.2процес починається.

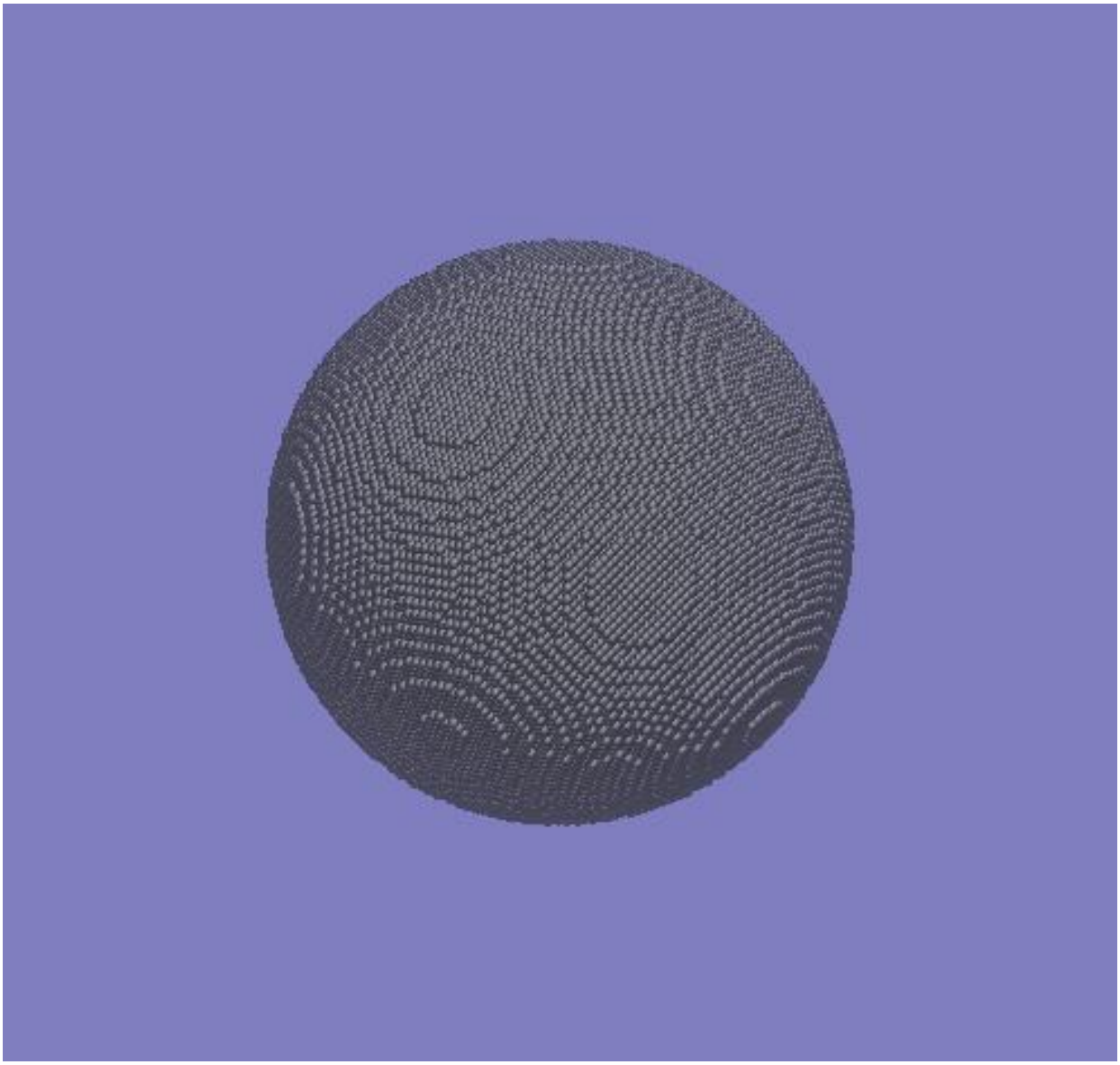
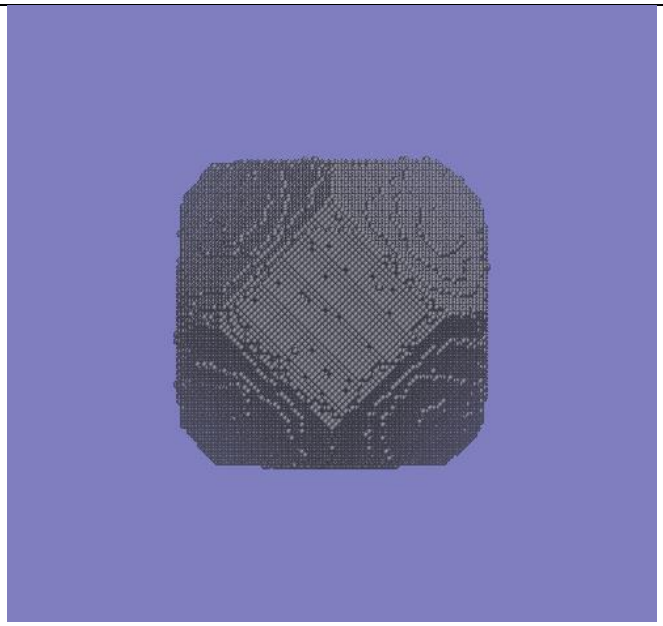


Рис. 3.2. Початкова конфігурація.

а)



б)

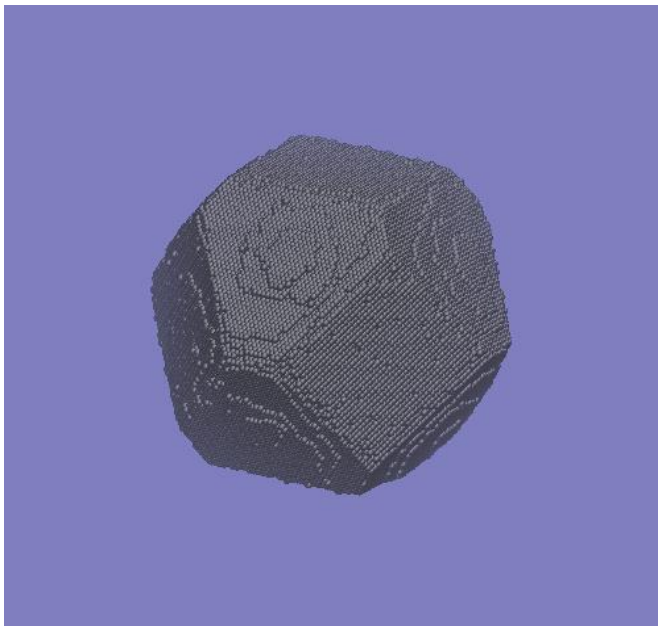


Рис. 3.3. Конфігурація на фінальному етапі експерименту

З плином кроків експерименту система набула сталої конфігурації (рис.3.3). Як можемо спостерігати вона співпадає з формую що набуває кластер в рівноважній конфігурації як вже було зазначено в підрозділі 3.3. З цього можемо судити про те, що модель була створена вдало.

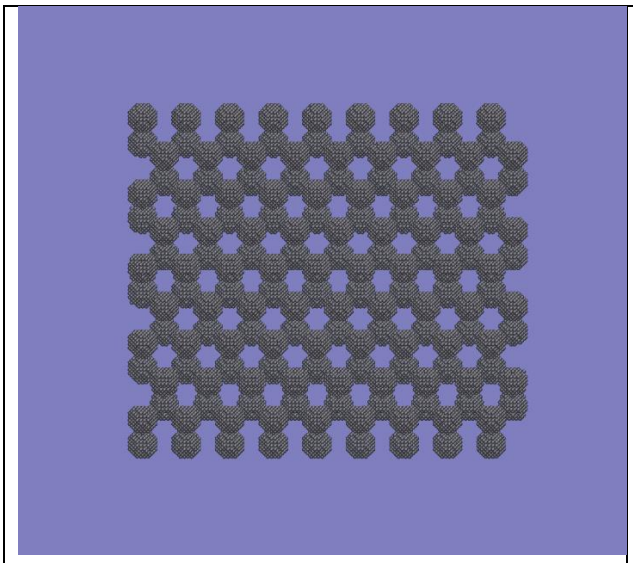
3.5. Експеримент №2: спікання металевих наночастинок

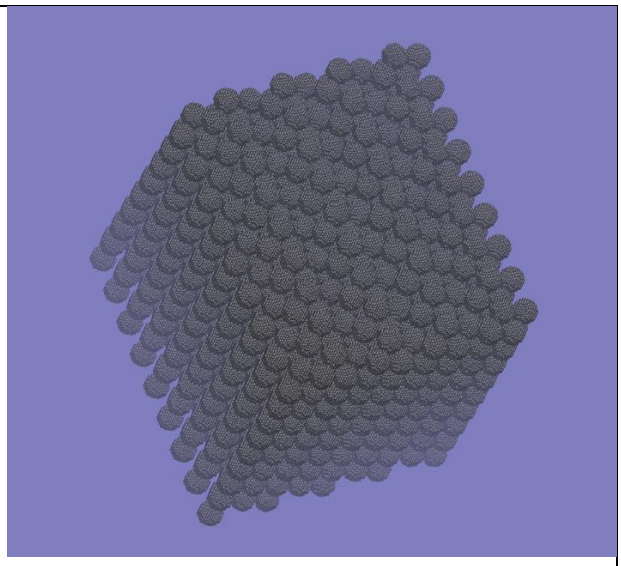
Наступним кроком було відтворення спікання ститсеми в трьохвимірному випадку. Перш за все постало питання створення початкової конфігурації для дослідження спікання системи в різчих часових та температурних проміжках. В основі створеної моделі було використано метод Монте-Карло.

За початкову конфігурацію було обрано сферичні кластери з гранецентрованою кубічною граткою. Котрі були розміщені в формі так званої «щільної упаковки»(гексагональної гратки).

Початкова конфігурація системи налічує 1531062 атоми. З яких утворено 1458 нанокластери. Нанокластери в моделі мають розмір порядку 3,2 нм. Буду вважати що дефектів в гратці немає. Нещодавні дослідження спікання двох нанокластерів, що були представлені[25], відображають спікання двох наноклстерів з розмірами порядку 4 нм. До того ж з контактом між ними. В створеній мною моделі нанокристали не мають контакту один з одним на початковій стадії. Відстань між кластерами порядку радіусу кластеру. На початковій стадії кластери – сфери з радіусом в 1,6 нм, що наповнені атомами, котрі впорядковані згідно гранецентрованої кристалічної гратки.

Суть проведених експериментів полягає в тому щоб знайти режим спікання, під час якого утворяться так звані перешийки з одного кластеру до декількох одночасно. Якщо утвориться лише один перешийок то з часом один з нанокластерів просто поглине інший.





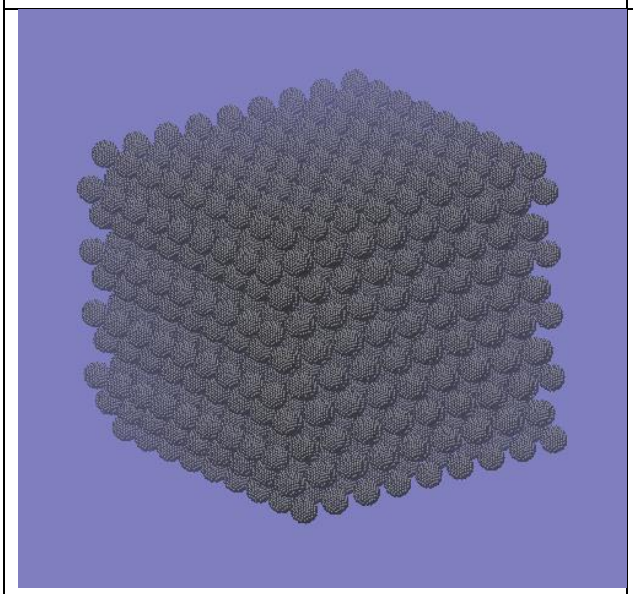


Рис. 3.4. Початкова конфігурація системи для проведення дослідів. Кластери з кубічною гранецентрованою кристалічною гратко. Розміщені в гексагональній гратці.

Наступним кроком було відтворення експериментів зі спіканням, температурні режими яких A07, A09, B07 ,B09(підрозділі 2.3). Для відображення процесу утворення перколяційного кластеру було використано алгоритм Хоше-Капельмана.

Режим A07.

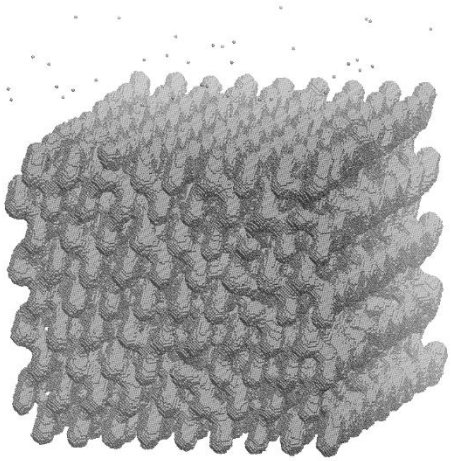


Рис. 3.5.1 Результат спікання початкової конфігурації в режимі A07.

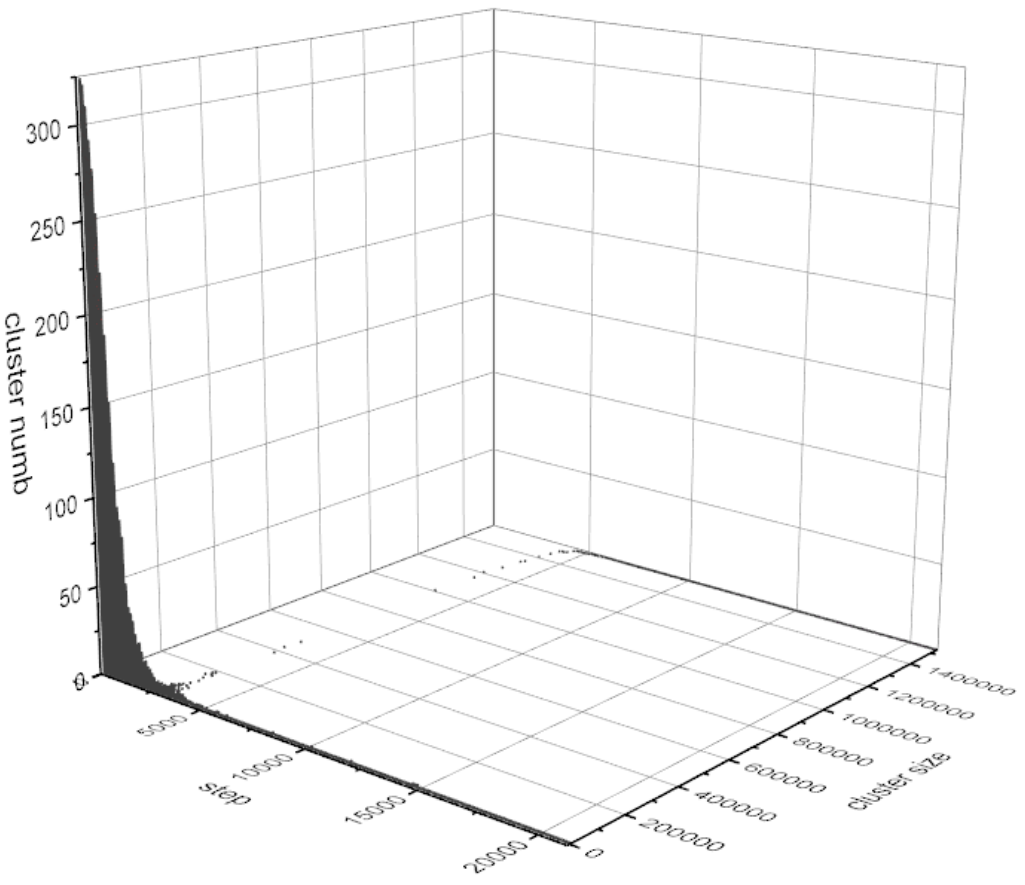


Рис. 3.5.2 Відображення прогресу утворення перколяційного кластеру в системі.

На рис 3.5.2 зображена динаміка утворення перколяційного кластеру. На початку маємо багато кластерів одноманітної форми, проте з часом їх стає меньше і вони стають частиною майбутнього перколяційного кластеру. Динаміку його утворення можна побачити вздовж осі cluster size.

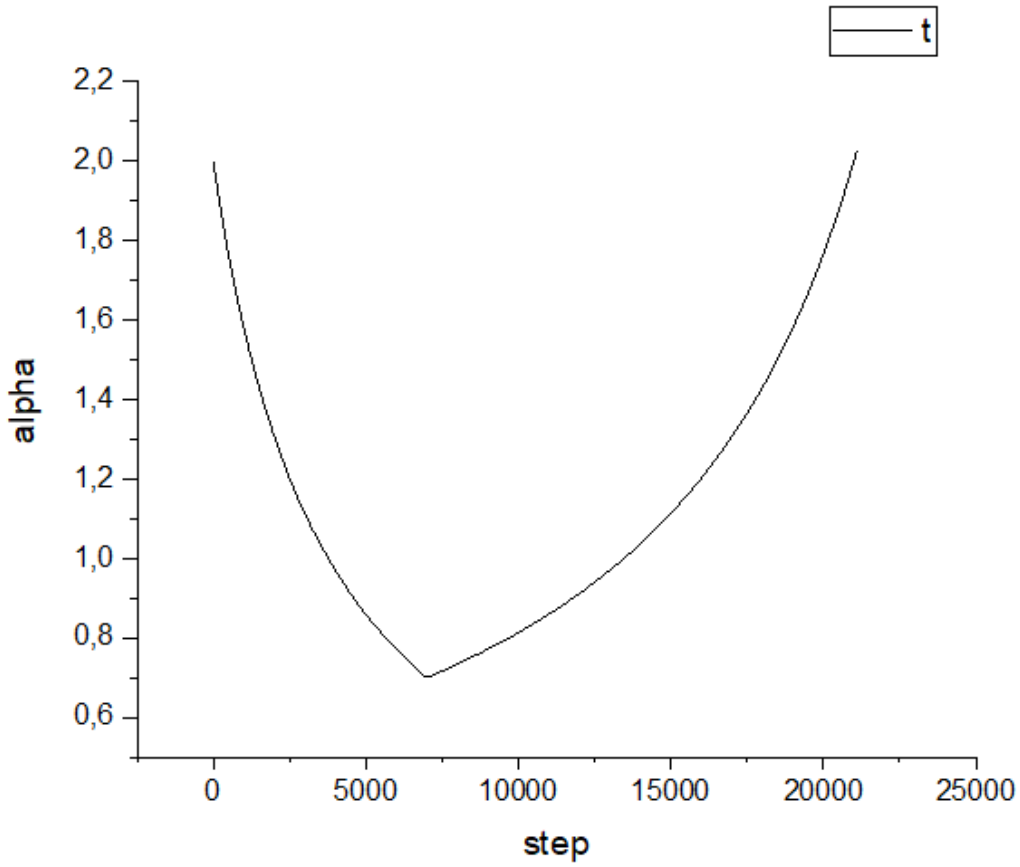


Рис. 3.5.3 Залежність коефіцієнту α, що пропорційний 1/T від кроків в експерименті

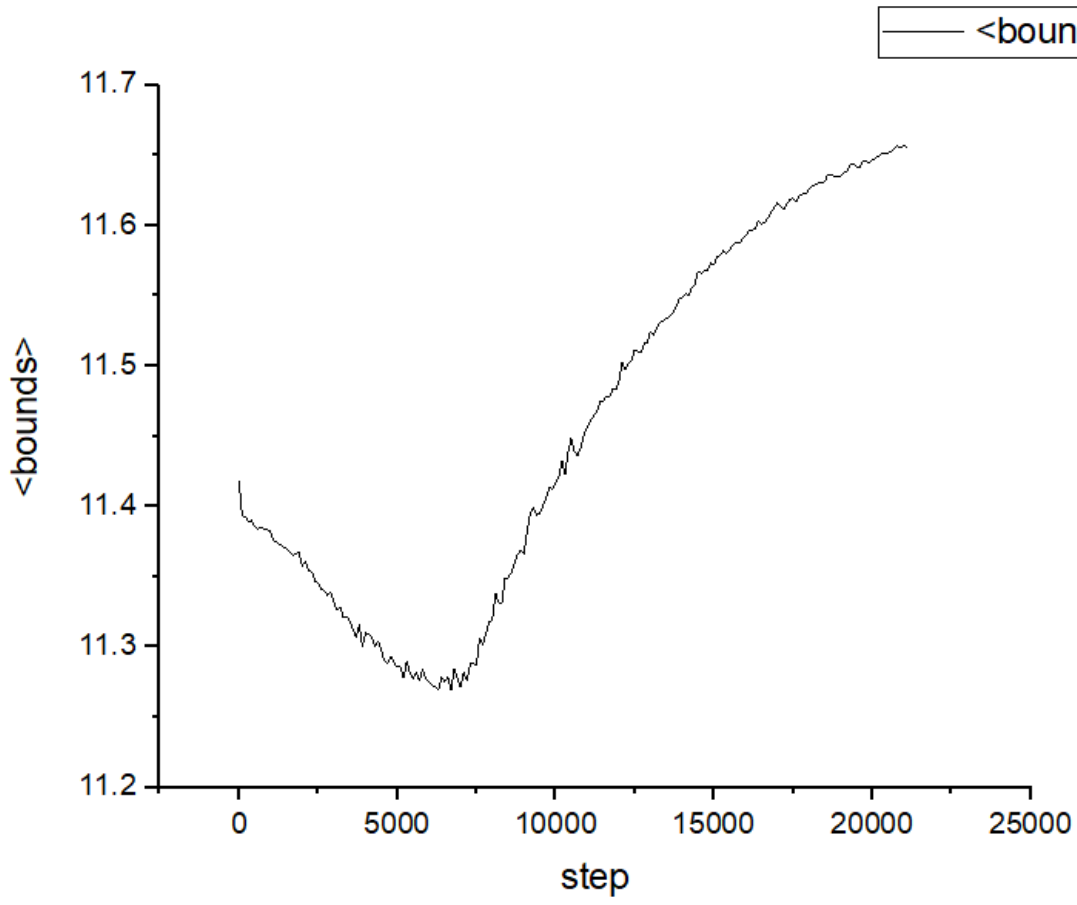


Рис. 3.5.4 Динаміка середнього число зв’язків атомів в системі.

Як ми можемо спостерігати в режимі A07 перколяційний кластер вже повністю сформований на 5000 кроці. Це можна побачити на графіку 3.5.2. Графік 3.5.3 вказує на те що це режим високотемпературного прогріву. З графіку, відображеного на рис 3.5.4, можна побачити початковий спад середньої кількості зв’язків атомів в системі. Це обумовлено Thermal Surface Roughening та утворенням місточків між кластерами. Надалі в силу вступає процес спікання. З точки, якої досяг графік (середня кількість зв’язків ~ 11,65) можемо прогнозувати високу механічну міцність утвореної конфігурації, оскільки стуктура буде мати багато тонких перемичок.

Режим A09.

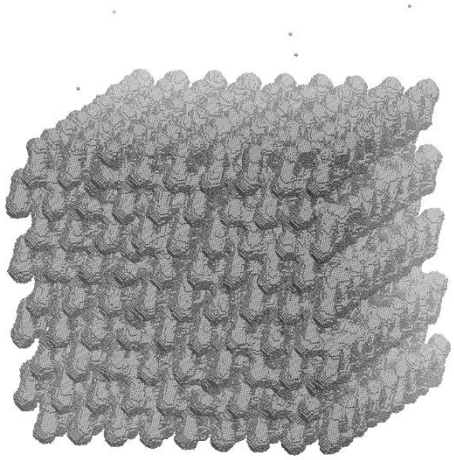


Рис. 3.6.1 Результат спікання початкової конфігурації в режимі A09.

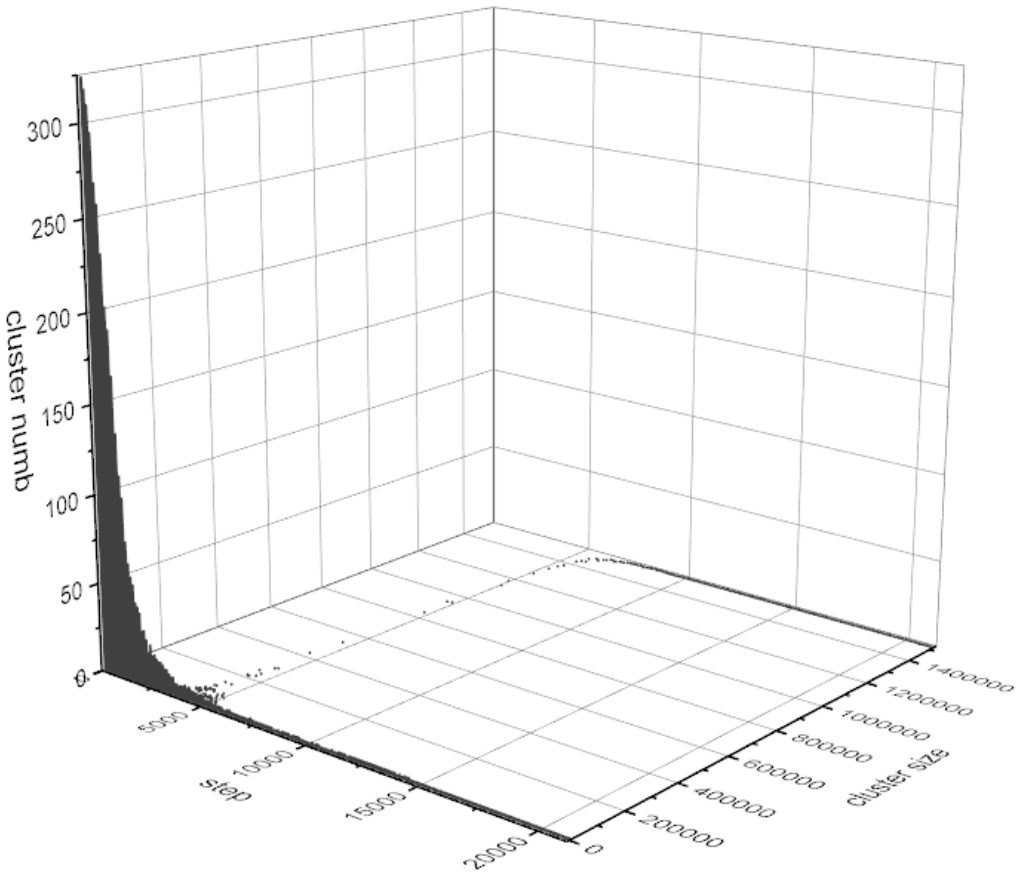


Рис. 3.6.2 Відображення прогресу утворення перколяційного кластеру в системі.

На рис 3.6.2 зображена динаміка утворення перколяційного кластеру. На початку маємо багато кластерів одноманітної форми, проте з часом їх стає меньше і вони стають частиною майбутнього перколяційного кластеру. Динаміку його утворення можна побачити вздовж осі cluster size.

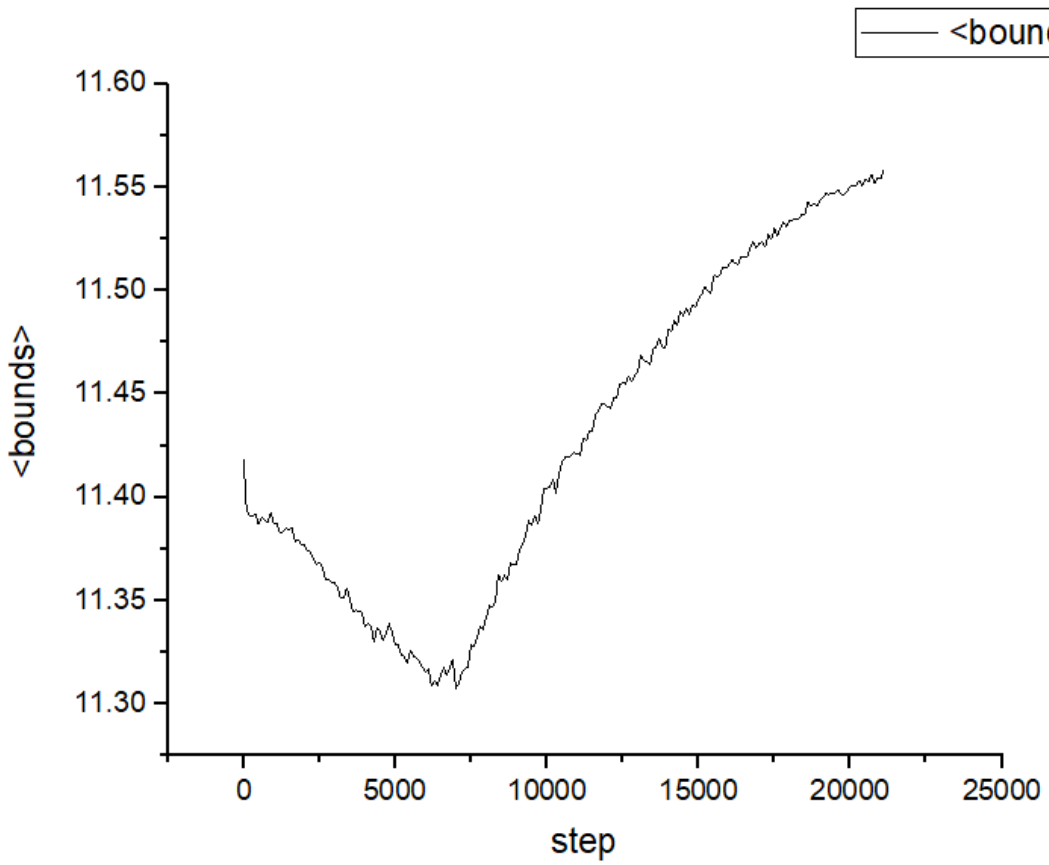


Рис. 3.6.3 Динаміка середнього число зв’язків атомів в системі.

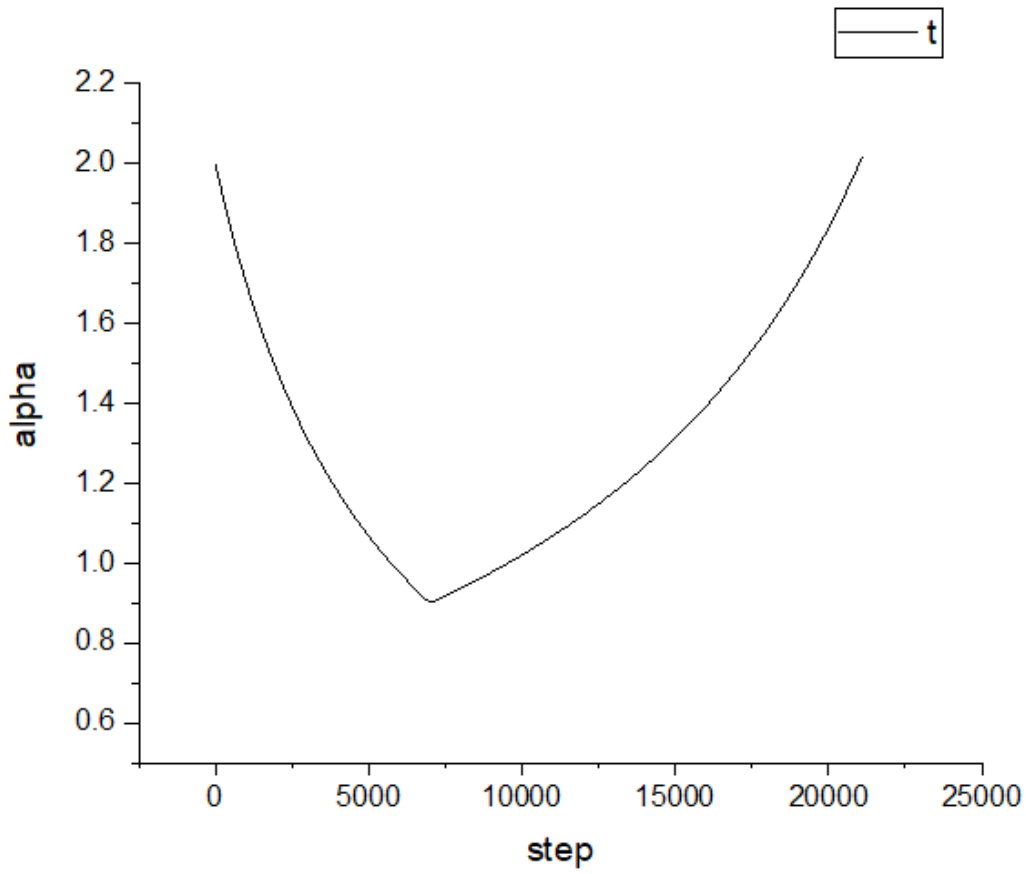


Рис. 3.6.4 Залежність коефіцієнту α, що пропорційний 1/T від кроків в експерименті

Як ми можемо спостерігати в режимі A09 перколяційний кластер вже повністю сформований на 7500 кроці. Це можна побачити на графіку 3.6.2. Графік 3.6.3 вказує на те що це режим так званого холодного прогріву. З графіку, відображеного на рис 3.6.4, можна побачити початковий спад середньої кількості зв'язків атомів в системі. Це обумовлено Thermal Surface Roughening та утворенням місточків між кластерами. Надалі в силу вступає процес спікання. З точки, якої досяг графік (середня кількість зв'язків ~ 11,56) можемо прогнозувати вищу механічну міцність утвореної конфігурації в порівнянні з режимом A07, оскільки стуктура буде мати більш тонші перемички.

Режим B07

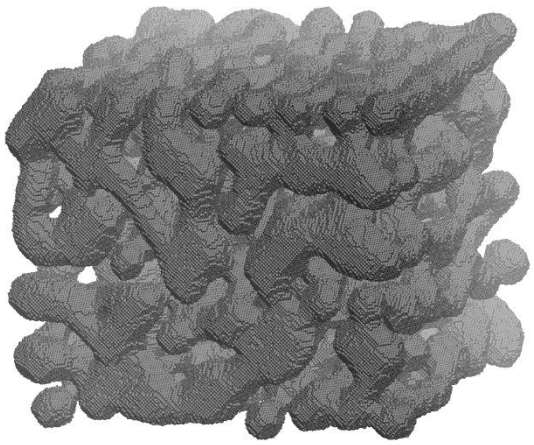


Рис. 3.7.1 Результат спікання початкової конфігурації в режимі B07.

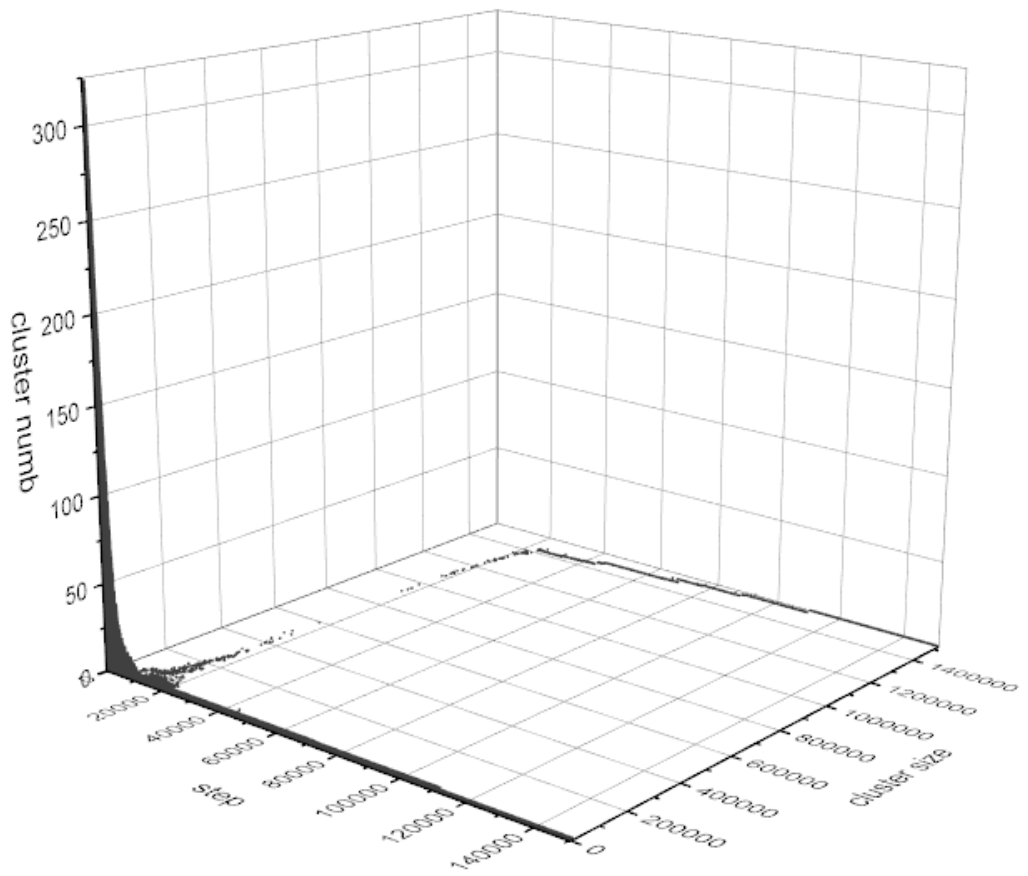


Рис. 3.7.2 Відображення прогресу утворення перколяційного кластеру в системі.

На рис 3.7.2 зображена динаміка утворення перколяційного кластеру. На початку маємо багато кластерів одноманітної форми, проте з часом їх стає меньше і вони стають частиною майбутнього перколяційного кластеру. Динаміку його утворення можна побачити вздовж осі cluster size.

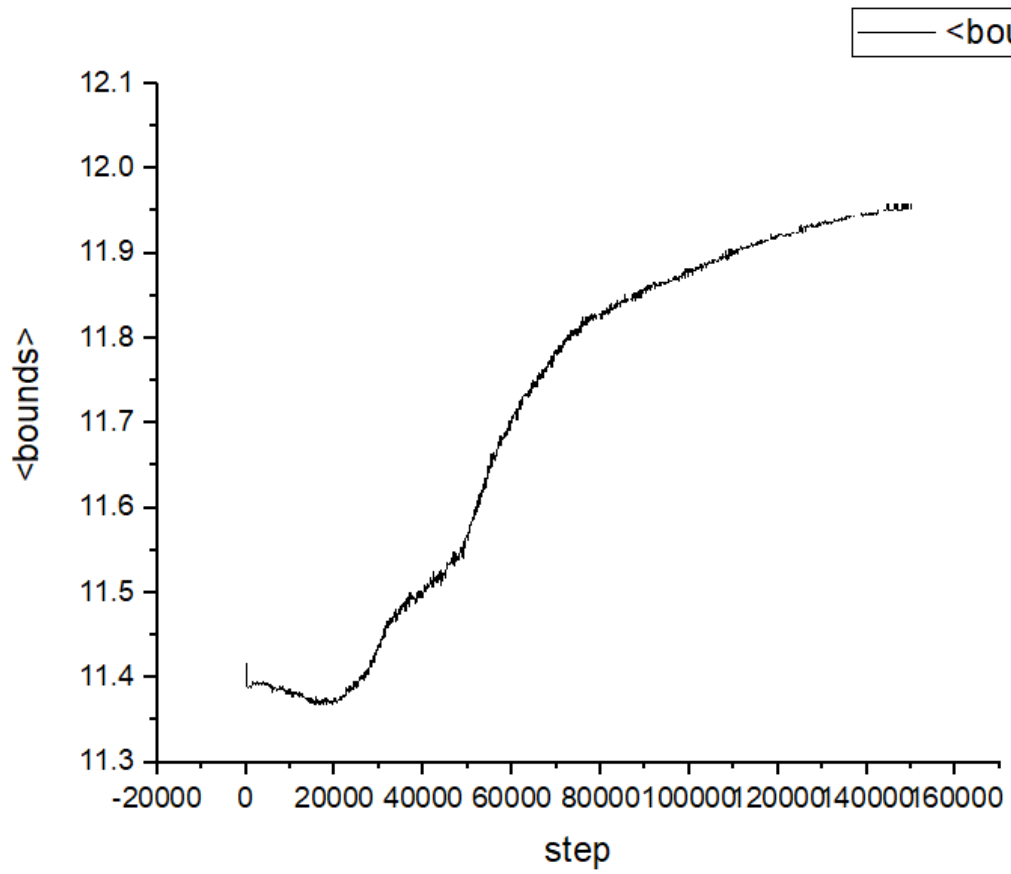


Рис. 3.7.3 Динаміка середнього число зв’язків атомів в системі.

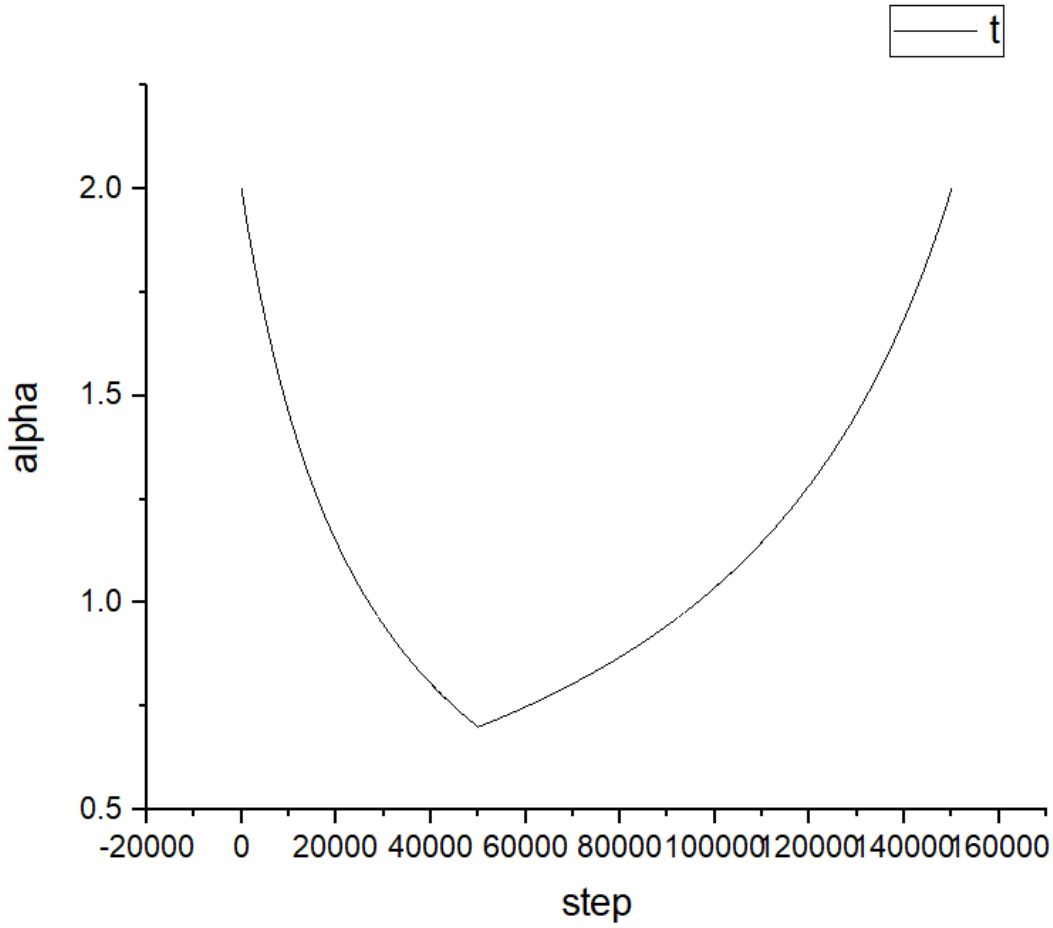


Рис. 3.7.4 Залежність коефіцієнту α, що пропорційний 1/T від кроків в експерименті

Як ми можемо спостерігати в режимі B07 перколяційний кластер вже повністю сформований на 22000 кроці. Це можна побачити на графіку 3.7.2. Графік 3.7.3 вказує на те що це режим високотемпературного прогріву. З графіку, відображеного на рис 3.7.4, можна побачити початковий спад середньої кількості зв'язків атомів в системі. Це обумовлено Thermal Surface Roughening та утворенням місточків між кластерами. Надалі в силу вступає процес спікання. З точки, якої досяг графік (середня кількість зв'язків ~ 11,96) можемо прогнозувати не таку високу механічну міцність утвореної конфігурації в порівнянні з режимами типу A, оскільки стуктура буде мати більш товщі перемички.

Режим B09

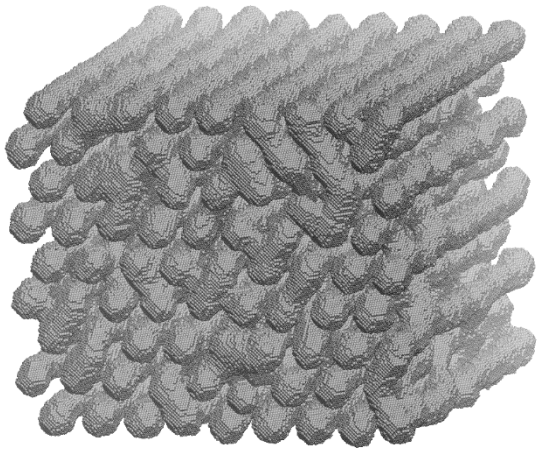


Рис. 3.8.1 Результат спікання початкової конфігурації в режимі B09.

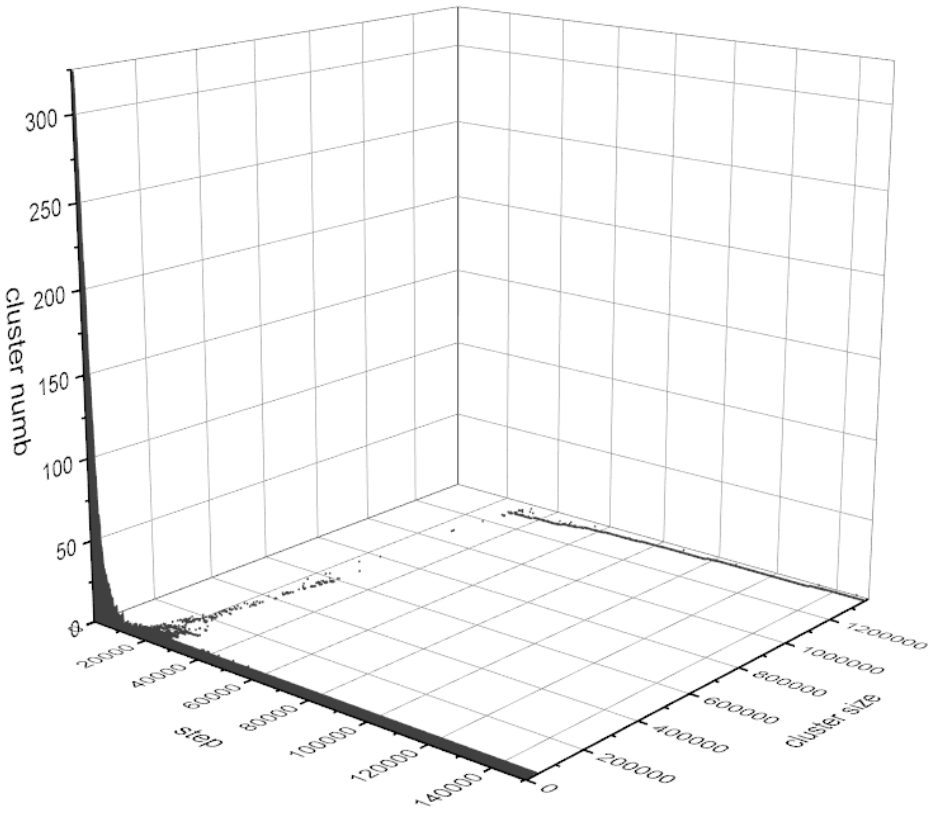


Рис. 3.8.2 Відображення прогресу утворення перколяційного кластеру в системі.

На рис 3.8.2 зображена динаміка утворення перколяційного кластеру. На початку маємо багато кластерів одноманітної форми, проте з часом їх стає меньше і вони стають частиною майбутнього перколяційного кластеру. Динаміку його утворення можна побачити вздовж осі cluster size.

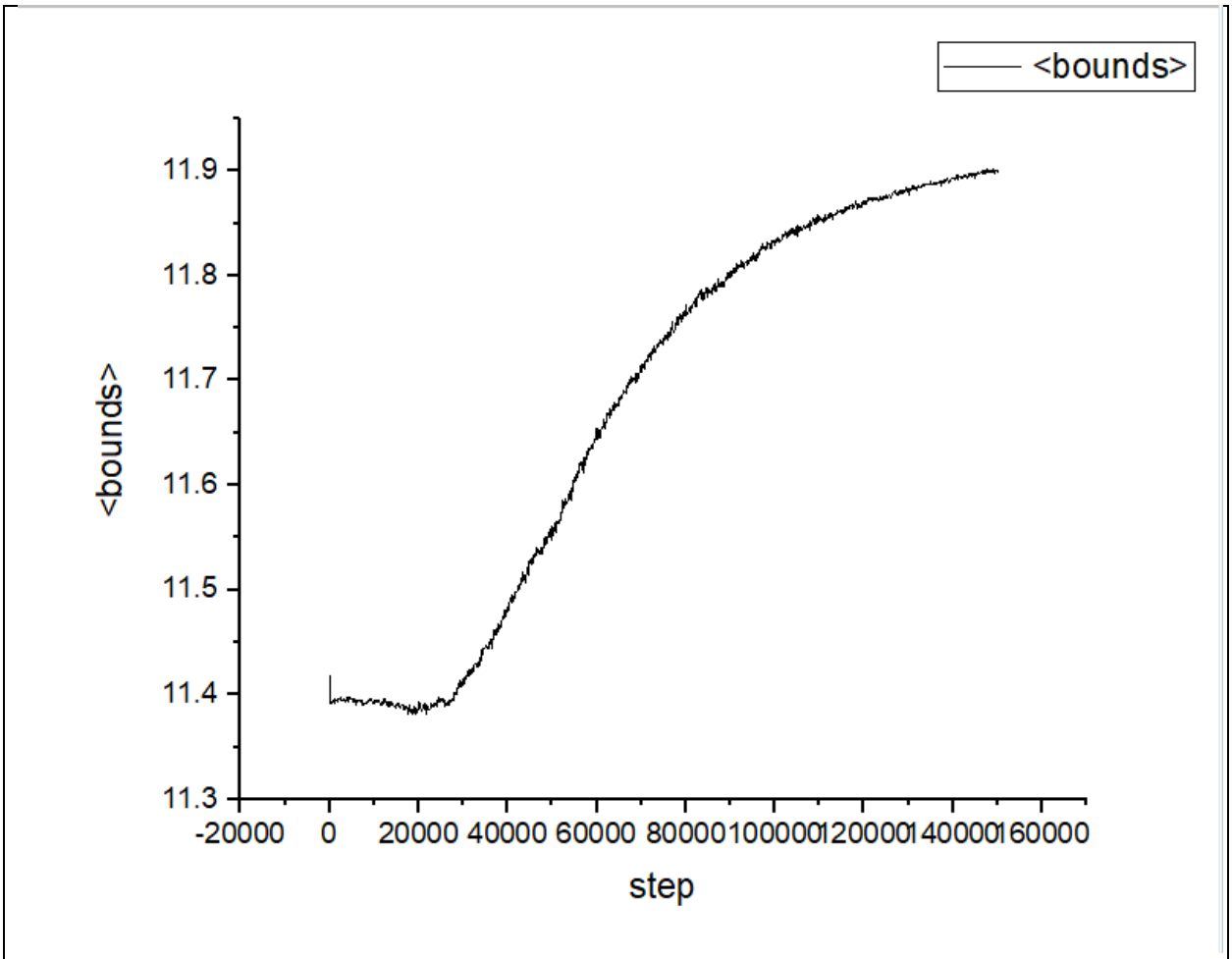


Рис. 3.8.3 Динаміка середнього число зв’язків атомів в системі.

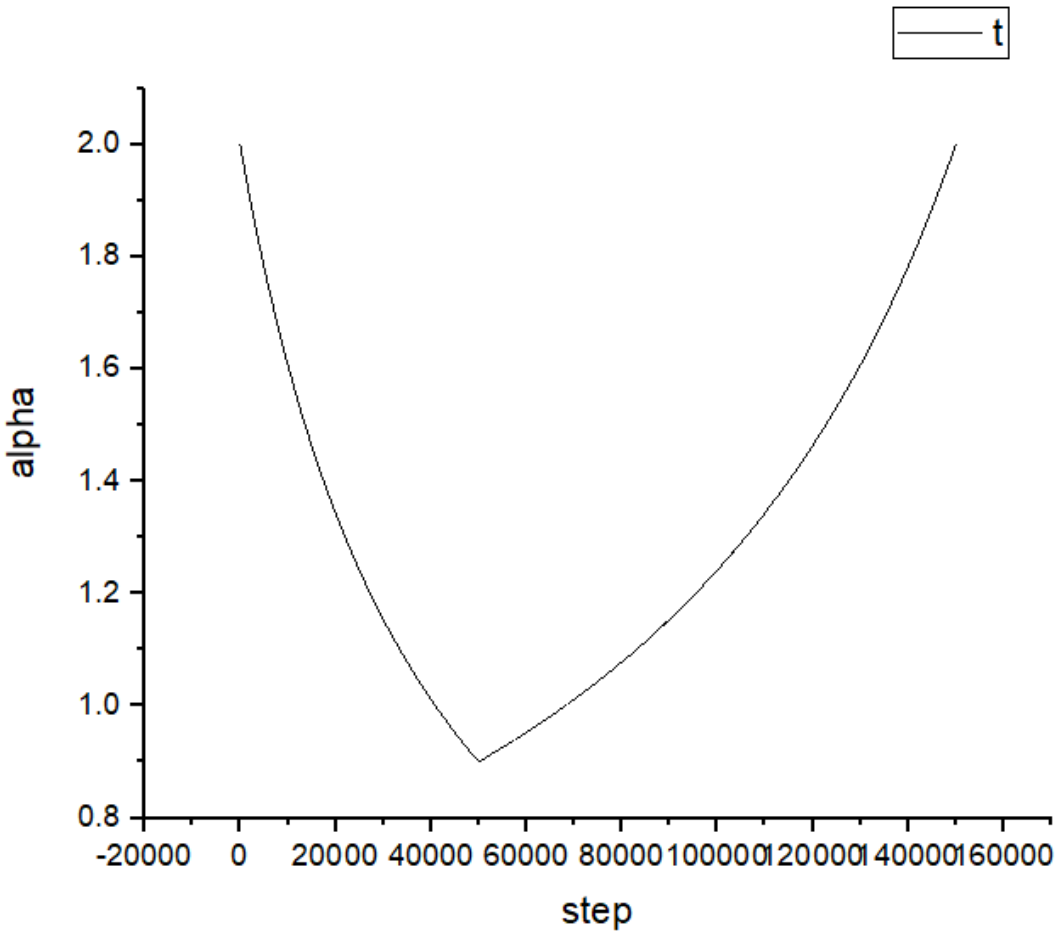


Рис. 3.8.4 Залежність коефіцієнту α, що пропорційний 1/T від кроків в експерименті

Як ми можемо спостерігати в режимі B09 перколяційний кластер вже повністю сформований на 30000 кроці. Це можна побачити на графіку 3.8.2. Графік 3.8.3 вказує на те що це режим так званого холодного прогріву. З графіку, відображеного на рис 3.8.4, можна побачити початковий спад середньої кількості зв'язків атомів в системі. Це обумовлено Thermal Surface Roughening та утворенням місточків між кластерами. Надалі в силу вступає процес спікання. З точки, якої досяг графік (середня кількість зв'язків ~ 11,88) можемо прогнозувати не таку високу механічну міцність утвореної конфігурації в порівнянні з режимами типу A, оскільки стуктура буде мати більш товщі перемички.

З рештою поставлених експериментів можна ознайомитись в додатку Б.

ВИСНОВКИ

При застосуванні до двовимірних шарів кінетична модель спікання призводить до декількох цікавих спостережень. Спостерігалося, що короткоімпульсний протокол нагріву до досить високої пікової температури є оптимальним для уникнення великих розривів у отриманій структурі. Розглянуті початкові розподіли розмірів частинок, які були обрані не надто полідисперсними, дозволяли проводити щільне початкове упакування. Однак не було очевидних переваг у пошуку по-справжньому монодисперсного (рівномірного) розподілу частинок за розмірами, оскільки якість спікання ще не покращилася. Також спостегрігалося, що спечені конфігурації мають певні особливості своєї морфології, які, очевидно, не зовсім локальні, але мають мезоскопічні розміри, значно більші, ніж малоатомні, і залежать від динаміки системи в цілому, а також від початкової конфігурації.

Дослідження в рамках цієї мезоскопічної кінетичної моделі МК включають тривимірні механізми частинок, обмежені лише кількома шарами частинок через необхідні обчислювальні ресурси.

У роботі представлена комп’ютерна модель спікання кластерів з кубічною гранецентрованою граткою. Врахована динаміка поверхневих частинок кластера, які з різним ступенем ймовірності можуть змінювати своє положення при переході в сусідні вакансії решітки або відриватися від поверхні. Проведені досліди підтвердили попередньо отримані результати в цій галузі й стали грутовними для подальшого дослідження якості структур, утворених за допомогою механізму спікання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Vyacheslav Gorshkov, Vasily Kuzmenko, Vladimir Privman J. Coupled Syst. Multiscale Dyn. August 2014 doi:10.1166/jcsmd.2014.1043
2. Vyacheslav Gorshkov, Vasily Kuzmenko, Vladimir Privman; Cryst. Eng. Comm doi:10.1039/c4ce01477d.
3. Y. Xia, Y. Xiong, B. Lim, and S. E. Skrabalak, Angew. Chem. 48, 60 (2009).
4. V. Privman, V. Gorshkov and O. Zavalov, Heat Mass Transfer, 2014, 50, 383.
5. F. Parhami and R. M. McMeeking, Mech. Mater., 1998, 27, 111.
6. Загальская Ю.Г., Литвинская Г.П., Егоров-Тисменко Ю.К. Геометрическая кристаллография. — М: Издательство Московского университета, 1986. — 168 с.
7. Фёдоров Е. С., Курс кристаллографии. Изд. 3-е, 1901 с. 51
8. Вайнштейн Б.К. Современная кристаллография. Том 1. Симметрия кристаллов, методы структурной кристаллографии. Наука, Москва, 1979.
9. Сиротин Ю.И., Шаскольская М.П. Основы кристаллофизики. Наука, Москва, 1979.
10. Флинт Е.Е. Практическое руководство по геометрической кристаллографии. Изд-е 3-е, перераб. и доп., Госгеолтехиздат, Москва, 1956.
11. Y. Cao, M. S. Denny, Jr., J. V. Caspar, W. E. Farneth, Q. Guo, A. S. Ionkin, L. K. Johnson, M. Lu, I. Malajovich, D. Radu, H. D. Rosenfeld, K. R. Choudhury, and W. Wu; J. Am. Chem. Soc. 134, 15644 (2012).
12. R. M. German; Liquid State Sintering, Plenum, New York (1985).
13. S. B. Rane, T. Seth, G. J. Phatak, D. P. Amalnerkar, and B. K. Das; Mater. Lett. 57, 3096 (2003).
14. H. Lippmann and R. Iankov; Int. J. Mech. Sci. 39, 585 (1997).
15. P. Bross and H. E. Exner; Acta Metal. 27, 1013 (1979).

16. V. Gorshkov and V. Privman; Physica E 43, 1 (2010). 43. I. Sevonkaev, V. Privman, and D. Goia; J. Solid State Electrochem. 17, 279 (2013). S Karim, M E Toimil-Molares, A G Balogh, W Ensinger, T WCornelius, E UKhan and R Neumann, Morphological evolution of Au nanowires controlled by Rayleigh instability, Nanotechnology 17 (2006) 5954–5959, doi:10.1088/0957-4484/17/24/009.

17. N. Combe, P. Jensen, and A. Pimpinelli, Phys. Rev. Lett.85, 110 (2000).

18. F. Wakai and F. Aldinger; Acta Materia. 51, 4013 (2003).

19.D. N. McCarthy and S. A. Brown, Phys. Rev. B 80, 064107 (2009).

20. R. Bjørk, H. L. Frandsen, N. Pryds, and V. Tikare; Scrip. Materia. 67, 81 (2012). V

21.. Gorshkov, V. Kuzmenko, and V. Privman; Cryst. Eng. Comm. 15, 7177 (2013).

22. V. Gorshkov, V. Kuzmenko and V. Privman, CrystEngComm, 2013, 15, 7177.

23. F. Wakai; J. Am. Ceram. Soc. 89, 1471 (2006).

24.V. N. Gorshkov and M. G. Chaban, Nonlinear electrohydrodynamic phenomena and droplet generation in charged jets of conducting liquid, Technical Physics, v. 44, Number 11 November 1999, pp. 1259-1266, doi:10.1134/1.1259506.

25. King C. Lai and James W. Evans PhysRevMaterials.3.026001 February 2019

26. Solids Far from Equilibrium, ed. C. Godrèche, Cambridge University Press, Cambridge, 1991.

27. Dynamics of Fractal Surfaces, ed. F. Family and T. Vicsek, World Scientific, Singapore, 1991.

28. J.-G. Oh, H.-S. Oh, W. H. Lee and H. Kim,J. Mater. Chem., 2012, 22, 15215.

29. R. M. German, Liquid State Sintering, Plenum, New York, 1985.

30. R. M. German, P. Suri and S. J. Park, J. Mater. Sci., 2009, 44, 1.

31. R. M. German, Int. J. Powder Metall., 2002, 38, 48.

32. V. Gorshkov, A. Zavalov and V. Privman, Langmuir, 2009, 25, 7940.

33. Matijevic, E. Colloid J. 2007, 69, 29.

34.A. Ruditskiy and Y. Xia, J. Am Chem. Soc. 138, 3161 (2016).

35. Nair, P. S.; Fritz, K. P.; Scholes, G. D. Small 2007, 3, 481.

36. Embden, van J.; Jasieniak, J.; Gomez, D. E.; Mulvaney, P.; Giersig, M. Aust. J. Chem. 2007, 60, 457.

37. Yin, Y. D.; Erdonmez, C.; Aloni, S.; Alivisatos, A. P. J. Am. Chem. Soc. 2006, 128, 12671.

38. Zeng, H. B.; Liu, P. S.; Cai, W. P.; Cao, X. L.; Yang, S. K. Cryst. Growth Des. 2007, 7, 1092.

39.B. P. Farrell, I. V. Sevonkaev and D. V. Goia, Platinum Met. Rev., 2013, 57, 161

40.A. P. Savitskii, Sci. Sintering, 2005, 37, 3.

41.M. Braginsky, V. Tikare and E. Olevsky, Int. J. Solids Struct., 2005, 42, 621.

42.Y. Cao, M. S. Denny Jr., J. V. Caspar, W. E. Farneth, Q. Guo, A. S. Ionkin, L. K. Johnson, M. Lu, I. Malajovich, D. Radu, H. D. Rosenfeld, K. R. Choudhury and W. Wu, J. Am. Chem. Soc., 2012, 134, 15644.

43.M. Vara, L. T. Roling, X. Wang, A. O. Elnabawy, Z. D. Hood, M.Chi,M.Mavrikakis,andY.Xia,ACSNano11,4571(2017).

ДОДАТОК А

Основний код програми.

```
public static void main(String[] args) {
   //Logger.INSTANCE.startLog();
   Grid grid = new Grid(301, 301, 301);
   GridHelper helper = new GridHelper();
   helper.grid = grid;
   helper.maxCoef = 1/1.5;
   helper.minCoef = 1/2.0;
   helper.steps = 5000;


   InitialConditionProvider conditionProvider = new
InitialConditionProvider(50, 100);
   conditionProvider.setupEnvironment(grid);


   MainWindow win = new MainWindow();
   win.setHelper(helper);
   win.setVisible(true);

}
package application;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;

import engine.ClusterData;
import engine.IntegrityDataCollector.LayerDataHandler;

public class Logger
{
    public final static Logger log = new Logger();

    private static final String ROOT_FOLDER = "Logs";
    private final String ExperimentDataContainer =
String.format("Instance_[%s]", getTimestamp());
    private final String snapshotFolderName = "Snapshots";
    private final String snapshotInvariantNamePart = "Snapshot_";
```

```
    private final String voidsFolderName = "VoidsData";
    private final String voidsInvariantNamePart = "VoidSnapshot_";

    private final String integrityLogFileName = "Integrity.txt";
    private final String strengthLogFileName = "Strength.txt";
    private final String tempereratureLogFileName = "Temperature.txt";
    private final String energyLogFileName = "Energy.txt";
    private final String LogFileName = "Log.txt";

    private BufferedWriter integrityLogWriter;
    private BufferedWriter strengthLogWriter;
    private BufferedWriter temperatureLogWriter;
    private BufferedWriter energyLogWriter;
    private BufferedWriter logWriter;


    protected Logger()
    {
       try {
         initializeLoggingSession();
      } catch (IOException e) {
         e.printStackTrace();
      }
    }

    private void initializeLoggingSession() throws IOException {
       File snapshotsFolder = new File(String.format("%s\\%s\\%s\\",
ROOT_FOLDER, ExperimentDataContainer, snapshotFolderName));
       snapshotsFolder.mkdirs();

       File voidsFolder = new File(String.format("%s\\%s\\%s\\", ROOT_FOLDER,
ExperimentDataContainer, voidsFolderName));
       voidsFolder.mkdirs();

       strengthLogWriter  = new BufferedWriter(new FileWriter(new
File(String.format("%s\\%s\\%s", ROOT_FOLDER, ExperimentDataContainer,
strengthLogFileName)), true));
       integrityLogWriter = new BufferedWriter(new FileWriter(new
File(String.format("%s\\%s\\%s", ROOT_FOLDER, ExperimentDataContainer,
integrityLogFileName)), true));
       temperatureLogWriter = new BufferedWriter(new FileWriter(new
File(String.format("%s\\%s\\%s", ROOT_FOLDER, ExperimentDataContainer,
tempereratureLogFileName)), true));
       energyLogWriter = new BufferedWriter(new FileWriter(new
File(String.format("%s\\%s\\%s", ROOT_FOLDER, ExperimentDataContainer,
energyLogFileName)), true));
       logWriter = new BufferedWriter(new FileWriter(new
File(String.format("%s\\%s\\%s", ROOT_FOLDER, ExperimentDataContainer,
LogFileName)), true));
    }


    public void println(String s) {
       String message = String.format("Time: %s <nanos: %s>|  %s",
getTimestamp(), System.nanoTime(),  s);
```

```
        System.out.println(message);
        try {
          logWriter.write(message);
          logWriter.newLine();
          logWriter.flush();
        } catch (Exception e) {
          System.err.println(String.format("Failed to append stream:
<logWriter> \r\n    %s", (Object)e.getStackTrace() ));
        }
    }

    public static String getTimestamp() {
        SimpleDateFormat sdfDate = new SimpleDateFormat("yyyy MM dd
(HH_mm_ss)");
        Date now = new Date();
        return sdfDate.format(now);
    }

    public void logSnapshot(Writable loggedOject) {
        Logger.log.println("Logging snapshot...");
        try {
            BufferedWriter writer = new BufferedWriter(new FileWriter(new
File(String.format("%s\\%s\\%s\\%s%s",
                    ROOT_FOLDER,
                    ExperimentDataContainer,
                    snapshotFolderName,
                    snapshotInvariantNamePart,
                    loggedOject.getName())), true));

            loggedOject.write(writer);
            writer.flush();
            writer.close();
        }
        catch(Exception e) {
            System.err.println(e.getMessage());
        }
        Logger.log.println("Logging snapshot finished");
    }

    public void logIntegritySnapshot(Writable loggedOject) {
        Logger.log.println("Logging integrity snapshot...");
        try {
            BufferedWriter writer = new BufferedWriter(new FileWriter(new
File(String.format("%s\\%s\\%s\\%s%s.txt",
                    ROOT_FOLDER,
                    ExperimentDataContainer,
                    voidsFolderName,
                    voidsInvariantNamePart,
                    loggedOject.getName())), true));

            loggedOject.write(writer);
            writer.flush();
            writer.close();
        }
        catch(Exception e) {
```

```
            System.err.println(e.getMessage());
        }
        Logger.log.println("Logging integrity snapshot finished");
    }

    public void logStrength(int step, double temp, ArrayList<ClusterData>
strengthData) {
        StringBuilder builder = new StringBuilder();

        for(ClusterData data : strengthData){
            builder.append(String.format("%s;", data.size()));
        }

        try {
            strengthLogWriter.write(String.format("{%-8d|%-20f| (%s)}\r\n",
step, temp, builder.toString()));
            strengthLogWriter.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }

        Logger.log.println("StrengthData data writing finished");
    }

    public void logIntegrity(int step, double temp, ArrayList<ClusterData>
integrityData) {
        StringBuilder builder = new StringBuilder();

        for(ClusterData data : integrityData){
            builder.append(String.format("%s;", data.size()));
        }

        try {
            integrityLogWriter.write(String.format("{%-8d|%-20f| (%s)}\r\n",
step, temp, builder.toString()));
            integrityLogWriter.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }

        Logger.log.println("Integrity data writing finished");
    }

    public void logTemperature(int step, double value) {
        try {
            temperatureLogWriter.write(String.format("%s | %s\r\n", step,
value));
            temperatureLogWriter.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
        finally {
            Logger.log.println(String.format("Step: %s; Temperature: %s", step,
value));
        }
```

```
    }

    public void logEnergy(int step, int[] distribution)  {

        StringBuilder b = new StringBuilder();
        for(int i: distribution) {
            b.append(i + " | ");
        }

        try {
            energyLogWriter.write(String.format("%s | %s\r\n", step,
b.toString()));
            energyLogWriter.flush();
       } catch (IOException e) {
          e.printStackTrace();
       }
        finally {
            Logger.log.println(String.format("Step: %s; Energy: %s", step,
b.toString()));
        }
    }

    public void logDebugSnapshot(Writable loggedOject) {
        Logger.log.println("Logging debug snapshot...");
        try {
            BufferedWriter writer = new BufferedWriter(new FileWriter(new
File(loggedOject.getName()), true));

            loggedOject.write(writer);

          writer.flush();
            writer.close();
        } catch (Exception e) {
          e.printStackTrace();
       }
        finally {
            Logger.log.println("Debug snapshot is logged");
        }
    }

    @Override
    public void finalize() {

        try {
            strengthLogWriter.flush();
            strengthLogWriter.close();
        }
        catch(Exception e) {

        }

        try {
            integrityLogWriter.flush();
            integrityLogWriter.close();
        }
```

```
        catch(Exception e) {

        }

      try {
         temperatureLogWriter.flush();
         temperatureLogWriter.close();
        }
        catch(Exception e) {

        }

      try {
         logWriter.flush();
         logWriter.close();
      }
      catch(Exception e) {

      }

    }

}

package application;

import java.util.ArrayList;

import Helpers.DPoint;
import Helpers.Point;
import engine.Grid;

public class InitialConditionProvider {

   private boolean[][][] validationPattern;
   int sphereRadius = 15, sphereDsitance = 10;



   public InitialConditionProvider() {

      validationPattern = new boolean[5][5][5];

      for(int i = 0; i < validationPattern.length; i++) {
         for(int j = 0; j < validationPattern[0].length; j++) {
            for(int k = 0; k < validationPattern[0][0].length; k++) {
               validationPattern[i][j][k] = false;
            }
         }
      }
      //hex grid interpretation
      /*
      validationPattern[0][0][0] = true;
      validationPattern[0][0][4] = true;
      validationPattern[0][4][0] = true;
```

```
        validationPattern[4][0][0] = true;
        validationPattern[0][4][4] = true;
        validationPattern[4][0][4] = true;
        validationPattern[4][4][0] = true;
        validationPattern[4][4][4] = true;

        validationPattern[2][2][0] = true;

        validationPattern[1][1][1] = true;
        validationPattern[3][3][1] = true;

        validationPattern[2][0][2] = true;
        validationPattern[0][2][2] = true;
        validationPattern[4][2][2] = true;
        validationPattern[2][4][2] = true;

        validationPattern[1][3][3] = true;
        validationPattern[3][1][3] = true;

        validationPattern[2][2][4] = true;
        */


        //square grid interpretation
        /*
        validationPattern[0][0][0] = true;

        validationPattern[2][0][0] = true;
        validationPattern[0][2][0] = true;
        validationPattern[0][0][2] = true;

        validationPattern[2][2][2] = true;

        validationPattern[0][2][2] = true;
        validationPattern[2][0][2] = true;
        validationPattern[2][2][0] = true;

        validationPattern[4][0][0] = true;
        validationPattern[0][4][0] = true;
        validationPattern[0][0][4] = true;

        validationPattern[0][4][4] = true;
        validationPattern[4][0][4] = true;
        validationPattern[4][4][0] = true;

        validationPattern[4][2][0] = true;
        validationPattern[4][0][2] = true;

        validationPattern[2][4][0] = true;
        validationPattern[2][0][4] = true;

        validationPattern[0][2][4] = true;
        validationPattern[0][4][2] = true;

        validationPattern[4][2][2] = true;
```

```
    validationPattern[4][4][2] = true;
    validationPattern[4][2][4] = true;
    validationPattern[2][4][4] = true;

    validationPattern[4][4][4] = true;


    validationPattern[1][1][1] = true;

    validationPattern[3][3][3] = true;

    validationPattern[3][1][1] = true;
    validationPattern[1][3][1] = true;
    validationPattern[1][1][3] = true;

    validationPattern[3][3][1] = true;
    validationPattern[3][1][3] = true;
    validationPattern[1][3][3] = true;
    */

    /////////
    validationPattern[0][0][0] = true;
    validationPattern[2][0][0] = true;
    validationPattern[4][0][0] = true;

    validationPattern[1][2][0] = true;
    validationPattern[3][2][0] = true;

    validationPattern[4][0][0] = true;
    validationPattern[4][2][0] = true;
    validationPattern[4][4][0] = true;

    /////////
    validationPattern[1][1][1] = true;
    validationPattern[3][1][1] = true;
    validationPattern[0][3][1] = true;
    validationPattern[2][3][1] = true;
    validationPattern[4][3][1] = true;

    /////////
    validationPattern[0][0][2] = true;
    validationPattern[2][0][2] = true;
    validationPattern[4][0][2] = true;

    validationPattern[1][2][2] = true;
    validationPattern[3][2][2] = true;

    validationPattern[4][0][2] = true;
    validationPattern[4][2][2] = true;
    validationPattern[4][4][2] = true;

    /////////
    validationPattern[1][1][3] = true;
    validationPattern[3][1][3] = true;
```

```
    validationPattern[0][3][3] = true;
    validationPattern[2][3][3] = true;
    validationPattern[4][3][3] = true;


    /////////
    validationPattern[0][0][4] = true;
    validationPattern[2][0][4] = true;
    validationPattern[4][0][4] = true;

    validationPattern[1][2][4] = true;
    validationPattern[3][2][4] = true;

    validationPattern[4][0][4] = true;
    validationPattern[4][2][4] = true;
    validationPattern[4][4][4] = true;



}

public InitialConditionProvider(int radius, int distance) {
    this();

    sphereRadius = radius;
    sphereDsitance = distance;
}

public void setupEnvironment(Grid grid) {

    DPoint center;
    double dx,dy,dz;
    dx = grid.dimX - 2 * sphereRadius - sphereDsitance;
    dy = grid.dimY - 2 * sphereRadius - sphereDsitance;
    dz = grid.dimZ - 2 * sphereRadius - sphereDsitance;

    ArrayList<DPoint> centers = new ArrayList<DPoint>();

    for(int i = 0; i < grid.dimX / sphereDsitance; i++) {
        for(int j = 0; j < grid.dimY / sphereDsitance; j++) {
            for(int k = 0; k < grid.dimZ / sphereDsitance; k++) {

                if(validationPattern[i % 4][j % 4][k % 4]
                    && i*sphereDsitance > 0 && i*sphereDsitance < dx
                    && j*sphereDsitance > 0 && j*sphereDsitance < dy
                    && k*sphereDsitance > 0 && k*sphereDsitance < dz
                )
                {
                    if(k % 2 ==0 ) {
                        centers.add(new DPoint(
                                (i + 0.5)*sphereDsitance,
                                (j + 0.5)*sphereDsitance * 0.866,
                                (k + 0.5)*sphereDsitance));
                    }
                    else {
```

```
                        centers.add(new DPoint(
                                (i + 0.5)*sphereDsitance,
                                ((j + 0.5)*sphereDsitance - sphereRadius / 2) *
0.866,
                                (k + 0.5)*sphereDsitance));
                    }
                }

            }
        }
    }

//    System.out.println(centers.size());

    for(int n = 0; n < centers.size(); n++) {

        for(int i = (int)(centers.get(n).x - sphereRadius); i <
(int)(centers.get(n).x + sphereRadius); i++) {
            for(int j = (int)(centers.get(n).y - sphereRadius); j <
(int)(centers.get(n).y + sphereRadius); j++) {
                for(int k = (int)(centers.get(n).z - sphereRadius); k <
(int)(centers.get(n).z + sphereRadius); k++) {

                    //sector = new Point(((i -
sphereDsitance/2)/sphereDsitance), ((j - sphereDsitance/2)/sphereDsitance),
((k - sphereDsitance / 2)/sphereDsitance));

                    center = centers.get(n);

                    if(grid.isValid(i, j, k)
                            && (center.x - i)*(center.x - i) + (center.y -
j)*(center.y - j) + (center.z - k)*(center.z - k) < sphereRadius*sphereRadius
                            ) {
                        grid.setPoint(i, j, k);
                    }

                }
            }
        }

    }



}
}
package application;

import java.awt.Graphics2D;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

import javax.swing.*;
```

```
import engine.GridHelper;

public class MainWindow extends JFrame{
   private JMenuBar menu;
   private JMenu mBegin, mFile;
   private JMenuItem miStartOne, miStartMany, miStop, miStartUpdates,
miStopUpdates, miCalculateAtoms,
                  miExportTxt, miExportVMD, miBeginExport, miEndExport,
miSave;
   private JLabel stepInfo, layerInfo;
   private Panel panel;

   public GridHelper helper;
   public boolean isUpdating;
// public ArrayList<IPaintable> paintedChildren;


   //Action section
   private ActionListener alStartOne = new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent arg0) {
         helper.startCalculation();
      }
   };

   private ActionListener alStartMany = new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent arg0) {
//       helper.startCalculation();
      }
   };

   private ActionListener alStop = new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent arg0) {
         helper.stopCalculation();
      }
   };

   private ActionListener alStartUpdates = new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent arg0) {
         startUpdates();
      }
   };

   private ActionListener alStopUpdates = new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent arg0) {
         stopUpdates();
      }
   };

   private  ActionListener alCalculateAtoms = new ActionListener() {
```

```
      @Override
      public void actionPerformed(ActionEvent arg0) {

      }
   };

   private ActionListener alExportTxt = new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent arg0) {
         try {

         } catch (Exception e) {
            e.printStackTrace();
         }
//       helper.saveForUser("");
      }
   };

   private ActionListener alExportVMD = new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent arg0) {
         try {
            helper.exportForVMD();
         } catch (Exception e) {
            e.printStackTrace();
         }
//       helper.saveForUser("");
      }
   };

   public MainWindow() {

      super();
      initialize();

      this.addKeyListener(new KeyListener() {

         @Override
         public void keyPressed(KeyEvent e) {
            //JOptionPane.showMessageDialog(null, e.getKeyCode());

            switch(e.getKeyCode()) {
               case(38):{
                  if(helper.currentlyPaintedPlane < helper.grid.dimZ - 1) {
                     helper.currentlyPaintedPlane++;
                  }
                  break;
               }
               case(40):{
                  if(helper.currentlyPaintedPlane > 0) {
                     helper.currentlyPaintedPlane--;
                  }
                  break;
               }
               case(107):{
```

```
                helper.setTemperature(helper.getTemperature() + 0.1);
                break;
             }
             case(109):{
                helper.setTemperature(helper.getTemperature() - 0.1);
                break;
             }
          }
          setTitle( Integer.toString( helper.currentlyPaintedPlane)  + "
" + helper.getTemperature());
       }

       @Override
       public void keyReleased(KeyEvent arg0) {
          // TODO Auto-generated method stub

       }

       @Override
       public void keyTyped(KeyEvent e) {

       }

    });
    isUpdating = false;
 }

 public void setHelper(GridHelper Helper) {
    helper = Helper;
    panel.addPainter(Helper);
 }

 public void initialize() {
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setSize(700, 500);
    setLocationRelativeTo(null);
    setExtendedState(MAXIMIZED_BOTH);
    setVisible(true);
    setLayout(null);

    panel = new Panel();
    panel.setBounds(0, 25, this.getBounds().width, this.getBounds().height
- 25);
    add(panel);

    menu = new JMenuBar();
    menu.setVisible(true);
    menu.setBounds(0, 0, this.getBounds().width, 25);
    //menu.setLayout(new FlowLayout());


    //mBegin section
    mBegin = new JMenu("Actions");
    //mBegin.setBounds(0, 0, 50, 20);
    menu.add(mBegin);
```

```
        miStartOne = new JMenuItem("Start one");
        miStartOne.addActionListener(alStartOne);
        mBegin.add(miStartOne);

        miStartMany = new JMenuItem("Start many");
        miStartMany.addActionListener(alStartMany);
        mBegin.add(miStartMany);

        miStop = new JMenuItem("Stop");
        miStop.addActionListener(alStop);
        mBegin.add(miStop);

        miStartUpdates = new JMenuItem("Start updates");
        miStartUpdates.addActionListener(alStartUpdates);
        mBegin.add(miStartUpdates);

        miStopUpdates = new JMenuItem("Stop updates");
        miStopUpdates.addActionListener(alStopUpdates);
        mBegin.add(miStopUpdates);

        miCalculateAtoms = new JMenuItem("Calculate atoms");
        miCalculateAtoms.addActionListener(alCalculateAtoms);
        mBegin.add(miCalculateAtoms);

        //end of mBegin section


        //mFile section

        mFile = new JMenu("File");
        //mBegin.setBounds(0, 0, 50, 20);
        menu.add(mFile);

        miExportTxt = new JMenuItem("Export to .txt file");
        miExportTxt.addActionListener(alExportTxt);
        mFile.add(miExportTxt);

        miExportVMD = new JMenuItem("Export for VMD");
        miExportVMD.addActionListener(alExportVMD);
        mFile.add(miExportVMD);

        miBeginExport =  new JMenuItem("Start Exporting");
//      miBeginExport.addActionListener(alStartVMDAutosaves);
        mFile.add(miBeginExport);

        miEndExport =  new JMenuItem("Stop Exporting");
//      miEndExport.addActionListener(alStopVMDAutosaves);
        mFile.add(miEndExport);

        miSave = new JMenuItem("Save as backup");
//      miSave.addActionListener(alSaveGrid);
        mFile.add(miSave);

        menu.add(mFile);
```

```
      add(menu);
      menu.updateUI();
   }

   public void startUpdates() {
      isUpdating = true;
      Thread t = new Thread() {
         public void run() {
            while(isUpdating) {
               panel.repaint();
               try {
                  Thread.currentThread().join(1);
               } catch (InterruptedException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
               }
            }
         }
      };
      t.start();
   }

   public void stopUpdates() {
      isUpdating = false;
   }


}
package application;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.util.ArrayList;

import javax.swing.JPanel;

import engine.IPaintable;

public class Panel extends JPanel{

   ArrayList<IPaintable> painters;

   public Panel() {
      setBackground(Color.WHITE);
      painters = new ArrayList<IPaintable>();
   }

   public void addPainter(IPaintable p) {
      painters.add(p);
   }
```

```
   public void paint(Graphics g2) {
      Graphics2D g = (Graphics2D) g2;

      g.setColor(Color.WHITE);
      g.fillRect(0, 0, 3000, 3000);

      g.setColor(Color.BLUE);

      for(int i = 0; i < painters.size(); i++) {
         painters.get(i).paint(g);
      }
   }

}
package application;

import java.io.BufferedWriter;

public interface Writable {

   public void write(BufferedWriter writer) throws Exception;
   public String getName();
   public void setName(String name);
}
package engine;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

import Helpers.Point;

public class ClusterData {
   private ArrayList<Point> incapsulatedMoleculas;

   public ClusterData()
   {
      incapsulatedMoleculas = new ArrayList<Point>();
   }

   public ClusterData(ArrayList<Point> particles)
   {
      incapsulatedMoleculas = particles;
   }

   public int size() {
      return incapsulatedMoleculas.size();
   }

}
package engine;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedList;
```

```java
import java.util.Random;

import Helpers.Point;
import application.Logger;

public class Grid {

   public byte[][][] grid;
   public ArrayList<Point> queue;
   public Random random;
   public int dimX, dimY, dimZ;
   public IntegrityDataCollector collector;
   //#region environment parameters


   public double alpha = 2;
   public double p0 = 0.685;

   //#endregion


   private boolean[][][] validationPattern;

   public Grid() {
      this(100, 100, 100);
   }

   public Grid(int DimX, int DimY, int DimZ) {
      dimX = DimX;
      dimY = DimY;
      dimZ = DimZ;

      validationPattern = new boolean[3][3][3];

      for(int i = 0; i < validationPattern.length; i++) {
         for(int j = 0; j < validationPattern[0].length; j++) {
            for(int k = 0; k < validationPattern[0][0].length; k++) {
               validationPattern[i][j][k] = false;
            }
         }
      }

      collector = new IntegrityDataCollector(this);

//    validationPattern[0][0][0] = true;
//    validationPattern[0][0][2] = true;
//    validationPattern[0][2][0] = true;
//    validationPattern[2][0][0] = true;
//    validationPattern[0][2][2] = true;
//    validationPattern[2][0][2] = true;
//    validationPattern[2][2][0] = true;
//    validationPattern[2][2][2] = true;
//
//    validationPattern[1][1][1] = true;
```

```
        validationPattern[0][0][0] = true;
        validationPattern[0][0][2] = true;
        validationPattern[0][2][0] = true;
        validationPattern[2][0][0] = true;
        validationPattern[0][2][2] = true;
        validationPattern[2][0][2] = true;
        validationPattern[2][2][0] = true;
        validationPattern[2][2][2] = true;

        validationPattern[0][1][1] = true;
        validationPattern[1][0][1] = true;
        validationPattern[1][1][0] = true;

        validationPattern[2][1][1] = true;
        validationPattern[1][2][1] = true;
        validationPattern[1][1][2] = true;


        grid = new byte[dimX][dimY][dimZ];
        queue = new ArrayList<Point>();
        random = new Random();
    }

    public void setAlpha(double value) {
        alpha = value;
        double ep0 = Math.pow(Math.E, alpha);
        p0 = Math.pow(0.685, alpha); // 0.685

        //Logger.INSTANCE.write(String .format("alpha: %s; p0: %s", alpha,
p0));

    }

    public double getAlpha() {
        return alpha;
    }

    //region

    //marks dimensions and grid specifics
    public boolean isValid(int x, int y, int z) {
        return(
                (x >= 0 && y >= 0 && z >= 0 && x < dimX && y < dimY && z < dimZ)
                && validationPattern[x % 2][y % 2][z % 2]
            );
    }

    public void setPoint(int i, int j, int k) {
        if(isValid(i, j, k)
            && grid[i][j][k] == 0)
        {
            grid[i][j][k] = 10;
            queue.add( new Point(i, j, k));
        }
    }
```

```
public void setPointUnchecked(int i, int j, int k) {
   if(isValid(i, j, k)) {
      grid[i][j][k] = 127;
      //queue.add(new Point(i, j, k));
   }
}

public int getNeirbourghsCount(int x, int y, int z) {
   int count = 0;

   for(int i = -1; i <= 1; i++ ) {
      for(int j = -1; j <= 1; j++ ) {
         for(int  k = -1; k <= 1; k++) {
            if(isValid(x + i, y + j, z + k)
                  && !(i == 0 && j == 0 && k == 0)
                  && grid[x + i][y + j][z + k] > 9) {
               count++;
            }
         }
      }
   }
   return count;
}

public boolean hasBoundedNeirbourghs(int x, int y, int z) {
   //boolean has = false;

   for(int i = -1; i <= 1; i++ ) {
      for(int j = -1; j <= 1; j++ ) {
         for(int  k = -1; k <= 1; k++) {
            if(isValid(x + i, y + j, z + k)
                  && !(i == 0 && j == 0 && k == 0)
                  && grid[x + i][y + j][z + k] > 9) {
               return true;
               /*has = true;
               break;*/
            }
         }
      }
   }
   return false;
}

/*public int getPositionWeight(int x, int y, int z) {

   int count = 0;

   for(int i = -1; i <= 1; i++ ) {
      for(int j = -1; j <= 1; j++ ) {
         for(int  k = -1; k <= 1; k++) {
            if(isValid(x + i, y + j, z + k)
                  && (i != 0 && j != 0 && k != 0)
                  && grid[x + i][y + j][z + k] > 9
                  && getNeirbourghsCount(x + i, y + j, z + k) > 0) {
```

```
                    count++;
                }
            }
        }
    }
    return count;
}*/

public ArrayList<PositionData> getBoundedPositionData(int x, int y, int
z){
    ArrayList<PositionData> data = new ArrayList<PositionData>();

    for(int i = -1; i <= 1; i++ ) {
        for(int j = -1; j <= 1; j++ ) {
            for(int  k = -1; k <= 1; k++) {

                if(isValid(x + i, y + j, z + k)
                        && !(i == 0 && j == 0 && k == 0)
                        && grid[x + i][y + j][z + k] == 0
                        ) {
                    data.add(new PositionData(x + i, y + j, z + k,
                            Math.exp( alpha * (getNeirbourghsCount(x + i, y + j,
z + k)))
                            ));
                }
            }
        }
    }

    data.add(new PositionData(x, y, z, Math.exp( alpha *
(getNeirbourghsCount(x, y, z)))));

    return data;
}

public ArrayList<PositionData> getFreePositionData(int x, int y, int z){
    ArrayList<PositionData> data = new ArrayList<PositionData>();

    for(int i = -1; i <= 1; i++ ) {
        for(int j = -1; j <= 1; j++ ) {
            for(int  k = -1; k <= 1; k++) {

                if(isValid(x + i, y + j, z + k)
                        && grid[x + i][y + j][z + k] == 0
                        ) {

                    data.add(new PositionData(x + i, y + j, z + k,
                            1
                            ));

                }
            }
        }
    }
```

```
      data.add(new PositionData(x, y, z, 1));

      return data;
   }

   public PositionData getNextPosition(int x, int y, int z) {
      ArrayList<PositionData> data;
//    if( grid[x][y][z] < 10) {
//       data = getFreePositionData(x, y, z);
//    }
//    else {
         data = getBoundedPositionData(x, y, z);
//    }

      double amplitude = 0;

      for(int i = 0; i < data.size(); i++) {
         amplitude += data.get(i).weight;
      }

      double point = amplitude * random.nextDouble();
      double sum = 0;

      for(int i = 0; i < data.size(); i++) {
         sum += data.get(i).weight;
         if(sum >= point) {
            return data.get(i);
         }
      }

      return data.get(data.size() - 1);
   }

   //from, where
   public void move(Point p, int i, int j, int k) {
      grid[p.x][p.y][p.z] = 0;
      /*boolean globalHandle = hasBoundedNeirbourghs(i, j, k);
      if(!globalHandle || isTopRegion(i, j, k)) {
         grid[i][j][k] = 1;
      }else {
         grid[i][j][k] = 10;
      }*/
      grid[i][j][k] = (byte) ((!hasBoundedNeirbourghs(i, j, k) ||
isTopRegion(i, j, k))?1:10);

      p.x = i;
      p.y = j;
      p.z = k;
   }

   public void jump(Point p) {
      int count = getNeirbourghsCount(p.x, p.y, p.z);
      if( count < 8 && random.nextDouble() < Math.pow(p0, count)/*Math.exp(-1
* p0 * count) */) {
         PositionData data = getNextPosition(p.x, p.y, p.z);
```

```
         if(grid[data.x][data.y][data.z] == 0) {
            move(p, data.x, data.y, data.z);
         }
      }
   }

   public void rearrange() {

      int x, y;
      Point temp;
      for(int i = 0; i < queue.size() / 2 ; i++) {
         x = random.nextInt(queue.size());
         y = random.nextInt(queue.size());

         temp = queue.get(x);
         queue.set(x, queue.get(y));
         queue.set(y, temp);

         //temp = null;
      }
   }

   //endregion

   public boolean isTopRegion(int i, int j, int k) {

      return k > dimZ;
   }

   public void refillTopRegion(double probability) {
      for(int i = 0; i < dimX; i++) {
         for(int j = 0; j < dimY; j++) {
            for(int k = 0; k < dimZ; k++) {
               if(isValid(i, j, k) && isTopRegion(i, j, k) && grid[i][j][k]
== 0 && random.nextDouble() < probability) {
                  setPoint(i, j, k);
               }
            }
         }
      }
   }

   public double probab() {
      double count = 0;
      double volume = 0;//(int) (dimX * dimY * dimZ * 0.2);


      for(int i = 0; i < dimX; i++) {
         for(int j = 0; j < dimY; j++) {
            for(int k = 0; k < dimZ; k++) {
               if(isTopRegion(i, j, k)) {
                  if(grid[i][j][k] != 0) {
                     count++;
                  }
                  if(isValid(i, j, k)) {
```

```
                    volume++;
                }
            }
        }
    }

    //System.out.println(count + " / " + volume + " count/volume");
    //0.015508
    return 0.001077 * 1.5 - count / volume;
//;//0.00128 * 4.0 - count / volume;//0.00017657 - count /
volume;//0.00009625 - count / volume;
  }

  public ArrayList<ClusterData> CollectClusterData() {
    ArrayList<ClusterData> data = new ArrayList<ClusterData>();

    synchronized(grid)
    {

      LinkedList<Point> particles = new LinkedList<Point>(this.queue);
      //byte[][][] volume = this.grid.clone();
      byte[][][] volume = new byte[dimX][dimY][dimZ];

      for(int i = 0; i < dimX; i++) {
        for(int j = 0; j < dimY; j++) {
          for(int k = 0; k < dimZ; k++) {
            volume[i][j][k] = grid[i][j][k];
          }
        }
      }

      //normalizing the array
      for(Point p : particles)
      {
        volume[p.x][p.y][p.z] = 1;
      }

      Point p;

      while(true)
      {
        //collect data for 1 cluster
        LinkedList<Point> q = new LinkedList<Point>();

        for(Point pp : particles) {
          if(volume[pp.x][pp.y][pp.z] == 1) {

            q.add(pp);
            volume[pp.x][pp.y][pp.z] = 10;

            break;
          }
        }
```

```
            if(q.size() == 0) break;

            for(int n = 0; n < q.size(); n++)
            {
               p = q.get(n);

               if(isValid(p.x, p.y, p.z))
               {
                  for(int i = -1; i <= 1; i++ ) {
                     for(int j = -1; j <= 1; j++ ) {
                        for(int  k = -1; k <= 1; k++) {
                           if(isValid(p.x + i, p.y + j, p.z + k)
                                 && volume[p.x + i][p.y + j][p.z + k] == 1)
                           {
                              Point p1 = new Point(p.x + i, p.y + j, p.z +
k);


                              q.add(p1);

                              volume[p1.x][p1.y][p1.z] = 10;
                           }
                        }
                     }
                  }
               }
               data.add(new ClusterData(new ArrayList<Point>(q)));
            }

         }

         return data;
      }

      public ArrayList<ClusterData> CollectinvertedClusterData() {
         ArrayList<ClusterData> data = new ArrayList<ClusterData>();

         synchronized(grid)
         {

            LinkedList<Point> particles = new LinkedList<Point>(this.queue);
            byte[][][] volume = new byte[dimX][dimY][dimZ];

            for(int i = 0; i < dimX; i++) {
               for(int j = 0; j < dimY; j++) {
                  for(int k = 0; k < dimZ; k++) {
                     if(isValid(i, j, k)) {
                        volume[i][j][k] = 1;
                     }
                  }
               }
            }

            //normalizing the array
            for(Point p : particles)
```

```
         {
            volume[p.x][p.y][p.z] = 0;
         }

         particles = new LinkedList<Point>();
         for(int i = 0; i < dimX; i++) {
            for(int j = 0; j < dimY; j++) {
               for(int k = 0; k < dimZ; k++) {
                  if(volume[i][j][k] == 1) {
                     particles.add(new Point(i, j, k));
                  }
               }
            }
         }

         Point p;

         while(true)
         {
            //collect data for 1 cluster
            LinkedList<Point> q = new LinkedList<Point>();

            for(Point pp : particles) {
               if(volume[pp.x][pp.y][pp.z] == 1) {

                  q.add(pp);
                  volume[pp.x][pp.y][pp.z] = 10;

                  break;
               }
            }

            if(q.size() == 0) break;

            for(int n = 0; n < q.size(); n++)
            {
               p = q.get(n);

               if(isValid(p.x, p.y, p.z))
               {
                  for(int i = -1; i <= 1; i++ ) {
                     for(int j = -1; j <= 1; j++ ) {
                        for(int  k = -1; k <= 1; k++) {
                           if(isValid(p.x + i, p.y + j, p.z + k)
                                 && volume[p.x + i][p.y + j][p.z + k] == 1)
                           {
                              Point p1 = new Point(p.x + i, p.y + j, p.z +
k);

                              q.add(p1);

                              volume[p1.x][p1.y][p1.z] = 10;
                           }
                        }
                     }
```

```
                    }
                }
            }
            data.add(new ClusterData(new ArrayList<Point>(q)));
        }
    }

    return data;
  }

  public double getAverageLinksCount() {
    int neirbourghs = 0;
    int particlesCount = 0;

    int tempNC = 0;

    for(Point p : queue) {
      tempNC = getNeirbourghsCount(p.x, p.y, p.z);
      if(tempNC < 8) {
        neirbourghs = neirbourghs + tempNC;
        particlesCount++;
      }
    }

    return neirbourghs/particlesCount;
  }

  public int[] getEnergyData() {
    int[] data = new int[14];
    int tempNC = 0;

    for(Point p : queue) {
      tempNC = getNeirbourghsCount(p.x, p.y, p.z);
      data[tempNC]++;
    }

    data[13] = queue.size();

    return data;
  }

}
package engine;

import java.awt.Color;
import java.awt.Graphics2D;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Date;

import Helpers.Point;
import application.Logger;
```

```
import application.Writable;
import engine.IntegrityDataCollector.Direction;
import engine.IntegrityDataCollector.LayerDataHandler;

public class GridHelper implements IPaintable{

   public Grid grid;
   public boolean calculationRunning;

   //#region paint settings

   public int currentlyPaintedPlane, halfSize = 2;
   public double minCoef = 0.4, maxCoef = 1; // minimum value of alpha in
temperature changing process
   public int steps = 5000;
   //#endregion


   public GridHelper() {
      calculationRunning = false;
      currentlyPaintedPlane = 0;
   }


   public void exportForVMD() throws IOException {
      Logger.log.logDebugSnapshot(new Writable() {

         private String name;

         @Override
         public void write(BufferedWriter writer) throws IOException {
            exportForVMD(writer);
         }

         @Override
         public String getName() {
            return "Debug.pdb";
         }

         @Override
         public void setName(String name) {
            this.name = name;
         }
      });
   }

   public void exportForVMD(BufferedWriter writer) throws IOException {
      synchronized(grid.grid) {

         for(Point p : grid.queue) {
            {
               writer.write("ATOM    100  N   VAL A  25     " +
(form(10*p.x)) + " " + (form(10*p.y)) + " " + (form(10*p.z)) + "  1.00 12.00
A1   C   " + System.lineSeparator());
            }
```

```
        }
    }
}

public String form(int number) {
    String s = Double.toString(number / 10.0);

    while (s.indexOf(".") < 3) {
        s = "0" + s;
    }

    while (s.length() < 7) {
        s += "0";
    }

    return s;
}

public void startCalculation() {
    calculationRunning = true;
    Thread t = new Thread() {
        public void run() {
            grid.rearrange();

            double size = grid.queue.size();
            Integer step = 0;

            Writable snapshot = new Writable() {

                private String name;

                @Override
                public void write(BufferedWriter writer) throws Exception {
                    exportForVMD(writer);
                }

                @Override
                public String getName() {
                    return name;
                }

                @Override
                public void setName(String name) {
                    this.name =  name;
                }
            };

            Writable voidDataX = new Writable() {

                private String name;

                @Override
                public void write(BufferedWriter writer) throws Exception {
```

```
                    IntegrityDataCollector collector = new
IntegrityDataCollector(grid);
                    ArrayList<LayerDataHandler> data =
collector.CollectPerLayerInvertedData(Direction.X);
                    for(int i = 0; i < data.size(); i++) {

                        StringBuilder builder = new StringBuilder();

                        for(int j = 0; j < data.get(i).data.size(); j++) {

                            builder.append(String.format("%s,
",data.get(i).data.get(j).size()));
                        }

                        writer.write(builder.substring(0,  builder.length() - 2)
+ "\r\n");
                    }

                }

                @Override
                public String getName() {
                    return name;
                }

                @Override
                public void setName(String name) {
                    this.name =  name;
                }
            };

            Writable voidDataY = new Writable() {

                private String name;

                @Override
                public void write(BufferedWriter writer) throws Exception {

                    IntegrityDataCollector collector = new
IntegrityDataCollector(grid);
                    ArrayList<LayerDataHandler> data =
collector.CollectPerLayerInvertedData(Direction.Y);
                    for(int i = 0; i < data.size(); i++) {

                        StringBuilder builder = new StringBuilder();

                        for(int j = 0; j < data.get(i).data.size(); j++) {

                            builder.append(String.format("%s,
",data.get(i).data.get(j).size()));
                        }

                        writer.write(builder.substring(0,  builder.length() - 2)
+ "\r\n");
                    }
```

```
            }

            @Override
            public String getName() {
               return name;
            }

            @Override
            public void setName(String name) {
               this.name =  name;
            }
         };

         Writable voidDataZ = new Writable() {

            private String name;

            @Override
            public void write(BufferedWriter writer) throws Exception {

               IntegrityDataCollector collector = new
IntegrityDataCollector(grid);
               ArrayList<LayerDataHandler> data =
collector.CollectPerLayerInvertedData(Direction.Z);
               for(int i = 0; i < data.size(); i++) {

                  StringBuilder builder = new StringBuilder();

                  for(int j = 0; j < data.get(i).data.size(); j++) {

                     builder.append(String.format("%s,
",data.get(i).data.get(j).size()));
                  }

                  writer.write(builder.substring(0,  builder.length() - 2)
+ "\r\n");
               }

            }

            @Override
            public String getName() {
               return name;
            }

            @Override
            public void setName(String name) {
               this.name =  name;
            }
         };


         while(calculationRunning) {
```

```
                    if(size < grid.queue.size()) {
                        size = grid.queue.size();
                        Logger.log.println(String.format("Particles were added!
Step: %s; CurrentCount: %s", step, grid.queue.size()));
                    }

//                  Logger.log.println(String.format("AverageLinksCount: %s",
grid.getAverageLinksCount()));

//                  grid.setAlpha(getTemperature(step));
//                  Logger.log.logTemperature(step, grid.alpha);

//                  ArrayList<ClusterData> data = grid.CollectClusterData();
//                  Logger.log.logIntegrity(step, grid.alpha, data);

                    snapshot.setName(String.format("%s.pdb",
Integer.toString(step)));
                    Logger.log.logSnapshot(snapshot);

                    Logger.log.logEnergy(step, grid.getEnergyData());

//                  voidDataX.setName("X_" + step);
//                  Logger.log.logIntegritySnapshot(voidDataX);
//
//                  voidDataY.setName("Y_" + step);
//                  Logger.log.logIntegritySnapshot(voidDataY);
//
//                  voidDataZ.setName("Z_" + step);
//                  Logger.log.logIntegritySnapshot(voidDataZ);


                    for(int j = 0; j < 100; j++) {
                        for(int i = 0; i < grid.queue.size(); i++) {
                            synchronized(grid.grid) {

grid.jump(grid.queue.get((grid.random.nextInt(grid.queue.size()))));
                            }
                        }
                    }

                    step += 100;

                    //стопор процесу для ПОТОЧНОГО графіку тмператури
//                  if(1/grid.alpha < minCoef) calculationRunning = false;
                }
            }
        };
        t.start();
    }

    public void stopCalculation(){
        calculationRunning = false;
    }

    public void rearrangeGridQueue() {
```

```
      if(grid != null) {
         int x, y; //future coordinates of elements
         Point temp;

         for(int i = 0; i < grid.queue.size(); i++) {
            x = grid.random.nextInt(grid.queue.size());
            y = grid.random.nextInt(grid.queue.size());

            temp = grid.queue.get(x);
            grid.queue.set(x, grid.queue.get(y));
            grid.queue.set(x, temp);
         }
      }
   }

   public double getTemperature() {
      return grid.alpha;
   }

   public void setTemperature(double d) {
      grid.setAlpha(d);
   }

   private double getTemperature(int step) {
      double value;

      if(step < steps)
      {
         value = 1 / ((maxCoef - minCoef)/steps * step + minCoef);
      }
      else {
         value = 1 / ( -0.5 * (maxCoef - minCoef)/steps * step + (3 * maxCoef
- minCoef)/2);
      }

      return value;
   }

   @Override
   public void paint(Graphics2D g) {
      g.setColor(Color.WHITE);
      g.fillRect(0, 0, 10000, 10000);



      g.setColor(Color.BLUE);

      for(int i = 0; i < grid.dimX; i++) {
         for(int j = 0; j < grid.dimY; j++) {

            if(grid.grid[i][j][currentlyPaintedPlane] != 0) {
//             g.setColor(Color.BLUE);
               g.fillRect( ( i) * halfSize, (  j) * halfSize, 1 * halfSize, 1
* halfSize);
//             g.fillOval( ( i) * halfSize, (  j) * halfSize, 1 * halfSize, 1
```

```
* halfSize);
            }
            else {
//             g.setColor(Color.GRAY);
//             g.fillRect( ( 2 *i) * halfSize + 1, ( 2* j) * halfSize + 1, 2
* halfSize - 2, 2 * halfSize - 2);
            }

         }
      }


   }

}
package engine;

import java.util.ArrayList;
import java.util.LinkedList;

import Helpers.Point;

public class IntegrityDataCollector {

   private byte[][][] sample;
   private Grid grid;

   public IntegrityDataCollector(Grid grid) {
      this.grid = grid;
   }

   public ArrayList<LayerDataHandler> CollectPerLayerData(Direction normal) {

      ArrayList<LayerDataHandler> data = new ArrayList<LayerDataHandler>();
      sample = copyGrid(grid.grid);

      int  limit = 0;
      switch(normal) {
         case X:{
            limit = grid.dimX;
            break;
         }

         case Y:{
            limit = grid.dimY;
            break;
         }

         case Z:{
            limit = grid.dimZ;
            break;
         }
      }

      for(int i = 0; i < limit; i++) {
```

```
            data.add(GetLayerData(i, normal));
        }

        return data;
    }

    public ArrayList<LayerDataHandler> CollectPerLayerInvertedData(Direction
normal) {

        ArrayList<LayerDataHandler> data = new ArrayList<LayerDataHandler>();
        sample = copyInvertedGrid(grid.grid);

        int  limit = 0;
        switch(normal) {
            case X:{
                limit = grid.dimX;
                break;
            }

            case Y:{
                limit = grid.dimY;
                break;
            }

            case Z:{
                limit = grid.dimZ;
                break;
            }
        }

        for(int i = 0; i < limit; i++) {
            data.add(GetLayerData(i, normal));
        }

        return data;
    }

    public LayerDataHandler GetLayerData(int layer, Direction normalDirection)
{
        LayerDataHandler layerData = new LayerDataHandler();
        layerData.layer = layer;
        layerData.normalDirection = normalDirection;

        ArrayList<Point> data;
        LinkedList<Point> queue;
        Point p;

        for(int i = normalDirection == Direction.X ? layer : 0; normalDirection
== Direction.X ? i == layer : i < grid.dimX; i = normalDirection ==
Direction.X ? grid.dimX : i + 1) {
            for(int j = normalDirection == Direction.Y ? layer : 0;
normalDirection == Direction.Y ? j == layer : j < grid.dimY; j =
normalDirection == Direction.Y ? grid.dimY : j + 1) {
                for(int k = normalDirection == Direction.Z ? layer : 0;
normalDirection == Direction.Z ? k == layer : k < grid.dimZ; k =
```

```
normalDirection == Direction.Z ? grid.dimZ : k + 1) {
               if(isValid(i, j, k) && sample[i][j][k] != 0)
               {
                  data = new ArrayList<Point>();
                  queue = new LinkedList<Point>();
                  queue.add(new Point(i, j, k));
                  sample[i][j][k] = 0;

                  while(queue.size() > 0) {
                     p = queue.removeFirst();

                     queue.addAll(GetLayerNeirbourghs(p.x, p.y, p.z,
normalDirection));

                     data.add(p);
                  }

                  layerData.data.add(new ClusterData(data));
               }
            }
         }
      }

      return layerData;
   }

   public LinkedList<Point> GetLayerNeirbourghs(int i, int j, int k,
Direction normalDirection) {
      LinkedList<Point> points = new LinkedList<Point>();

      for(int x = normalDirection == Direction.X ? 0 : -1; normalDirection ==
Direction.X ? x == 0 : x <= 1; x++) {
         for(int y = normalDirection == Direction.Y ? 0 : -1; normalDirection
== Direction.Y ? y == 0 : y <= 1; y++) {
            for(int z = normalDirection == Direction.Z ? 0 : -1;
normalDirection == Direction.Z ? z == 0 : z <= 1; z++) {

               if( !(x==0 && y == 0 && z == 0)
                  && isValid(i + x, j + y, z + k)
                  && sample[i + x][j + y][z + k] != 0)
               {
                  points.add(new Point(i + x, j + y, z + k));
                  sample[i + x][j + y][z + k] = 0;
               }
            }
         }
      }

      return points;
   }

   public boolean isValid(int i, int j, int k) {
      return grid.isValid(i, j, k);
   }
```

```
public ArrayList<ClusterData> CollectInvertedClusterData() {
   ArrayList<ClusterData> data = new ArrayList<ClusterData>();
   /*
   synchronized(grid)
   {

      LinkedList<Point> particles = new LinkedList<Point>(this.queue);
      byte[][][] volume = new byte[dimX][dimY][dimZ];

      for(int i = 0; i < dimX; i++) {
         for(int j = 0; j < dimY; j++) {
            for(int k = 0; k < dimZ; k++) {
               if(isValid(i, j, k)) {
                  volume[i][j][k] = 1;
               }
            }
         }
      }

      //normalizing the array
      for(Point p : particles)
      {
         volume[p.x][p.y][p.z] = 0;
      }

      particles = new LinkedList<Point>();
      for(int i = 0; i < dimX; i++) {
         for(int j = 0; j < dimY; j++) {
            for(int k = 0; k < dimZ; k++) {
               if(volume[i][j][k] == 1) {
                  particles.add(new Point(i, j, k));
               }
            }
         }
      }

      Point p;

      while(true)
      {
         //collect data for 1 cluster
         LinkedList<Point> q = new LinkedList<Point>();

         for(Point pp : particles) {
            if(volume[pp.x][pp.y][pp.z] == 1) {

               q.add(pp);
               volume[pp.x][pp.y][pp.z] = 10;

               break;
            }
         }

         if(q.size() == 0) break;
```

```
            for(int n = 0; n < q.size(); n++)
            {
               p = q.get(n);

               if(isValid(p.x, p.y, p.z))
               {
                  for(int i = -1; i <= 1; i++ ) {
                     for(int j = -1; j <= 1; j++ ) {
                        for(int  k = -1; k <= 1; k++) {
                           if(isValid(p.x + i, p.y + j, p.z + k)
                                 && volume[p.x + i][p.y + j][p.z + k] == 1)
                           {
                              Point p1 = new Point(p.x + i, p.y + j, p.z +
k);

                              q.add(p1);

                              volume[p1.x][p1.y][p1.z] = 10;
                           }
                        }
                     }
                  }
               }
               data.add(new ClusterData(new ArrayList<Point>(q)));
            }
         }
         */
         return data;
      }

      public byte[][][] copyGrid(byte[][][] g)
      {
         byte[][][] res = new byte[grid.dimX][grid.dimY][grid.dimZ];

         for(int i = 0; i < grid.dimX; i++) {
            for(int j = 0; j < grid.dimY; j++) {
               for(int k = 0; k < grid.dimZ; k++) {
                  if(g[i][j][k] != 0) {
                     res[i][j][k] = 1;
                  }
               }
            }
         }

         return res;
      }

      public byte[][][] copyInvertedGrid(byte[][][] g)
      {
         byte[][][] res = new byte[grid.dimX][grid.dimY][grid.dimZ];

         for(int i = 0; i < grid.dimX; i++) {
            for(int j = 0; j < grid.dimY; j++) {
               for(int k = 0; k < grid.dimZ; k++) {
```

```
                if(g[i][j][k] == 0) {
                    res[i][j][k] = 1;
                }
                else {
                    res[i][j][k] = 0;
                }
            }
        }
    }

    return res;
}


public enum Direction{
    X,
    Y,
    Z;
}

public class LayerDataHandler{
    public int layer;
    public Direction normalDirection;
    public ArrayList<ClusterData> data;

    public LayerDataHandler() {
        layer = 0;
        normalDirection = Direction.X;
        data = new ArrayList<ClusterData>();
    }
}

}
```

ДОДАТОК Б

Також було поставлено додаткові експерименти зі спіканням.

Частина перша: експерименти зі спукання з початковою конфігурацією, яка була зазначена в підрозділі 3.5. Експерименти тривали 150000 кроків. Причому α лінійно змінювався від 2,5 до α min за перші 50000 кроків, і повертався в початковий стан за 100000 корків. Далі буде представлено результати спукання при різних α min.

α min = 1,0

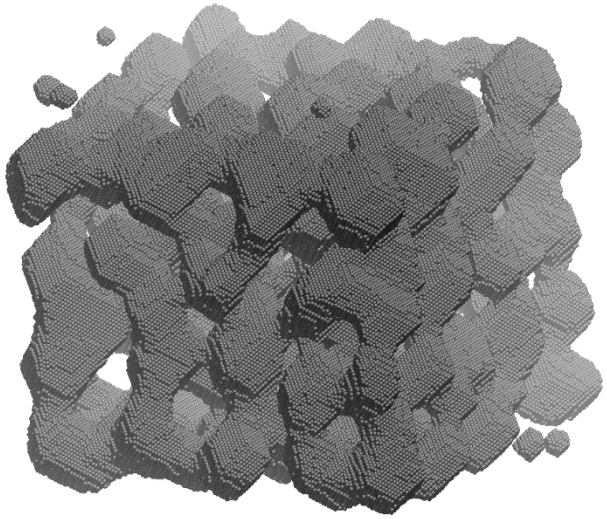


Рис. 1. Результат спікання нанокластерів металу за динаміки α *2,5->(50000 кроків)->1,0->(100000 кроків)->2,5*

α min = 1,25

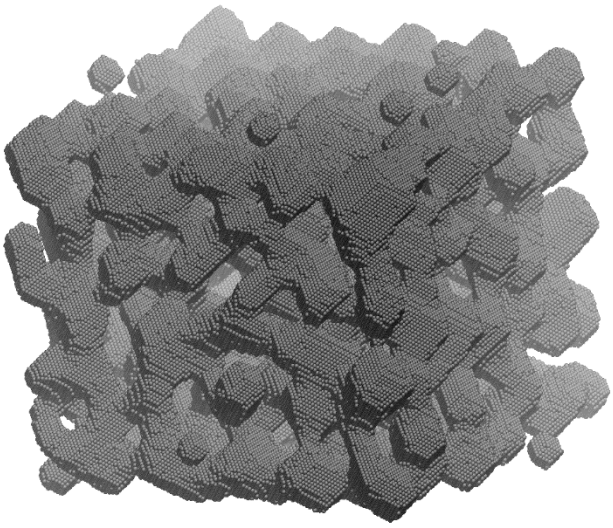


Рис. 2. Результат спікання нанокластерів металу за динаміки α *2,5->(50000 кроків)->1,25->(100000 кроків)->2,5*

α min = 1,4

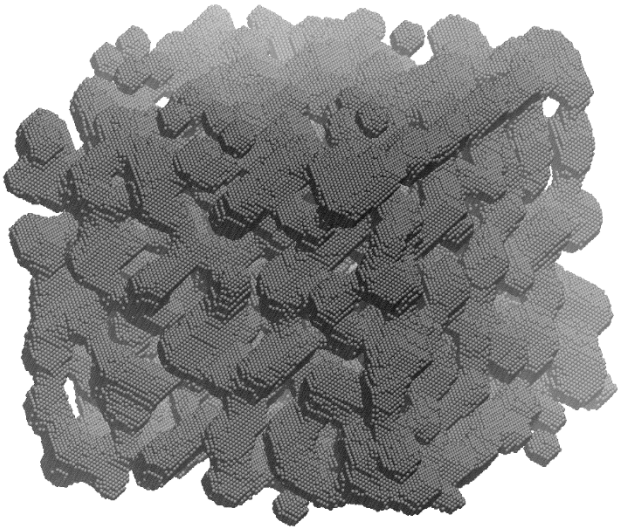


Рис. 3. Результат спікання нанокластерів металу за динаміки α *2,5->(50000 кроків)->1,4->(100000 кроків)->2,5*

α min = 1,5

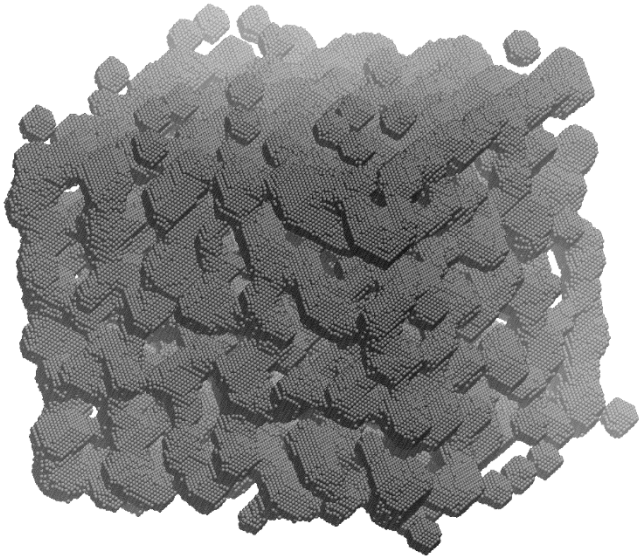


Рис. 4. Результат спікання нанокластерів металу за динаміки α *2,5->(50000 кроків)->1,5->(100000 кроків)->2,5*

α min = 1,6

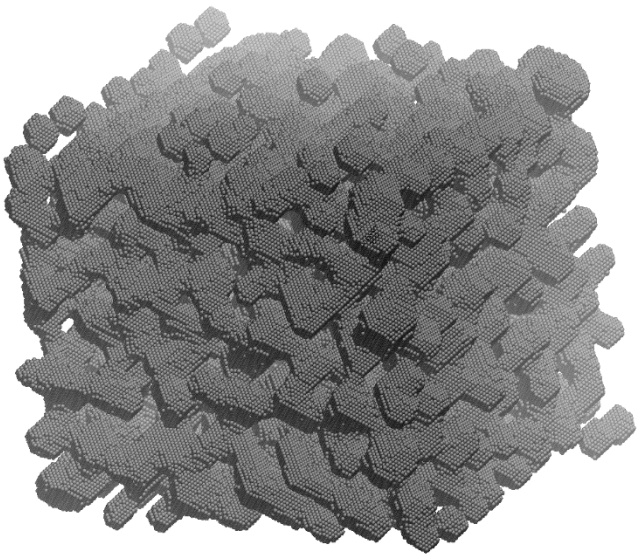


Рис. 5. Результат спікання нанокластерів металу за динаміки α *2,5->(50000 кроків)->1,6->(100000 кроків)->2,5*

Частина друга: експерименти зі спукання з початковою конфігурацією, яка була зазначена в підрозділі 3.5. Експерименти тривали 150000 кроків. Причому α лінійно змінювався від 2 до α min за перші 7000 кроків, і повертався в початковий стан за 14000 корків. Далі буде представлено результати спукання при різних α min.

α min = 0,6

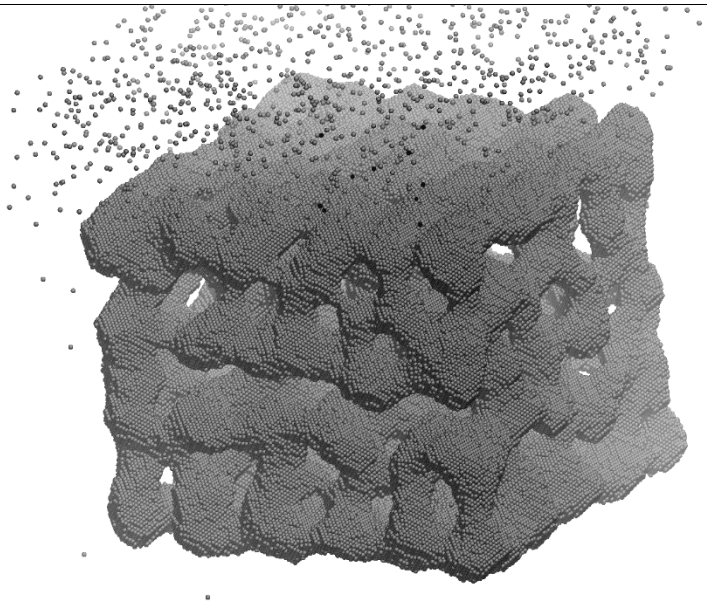


Рис. 6. Результат спікання нанокластерів металу за динаміки α *2,0->(7000 кроків)->0,6->(14000 кроків)->2,0*

α min = 0,7

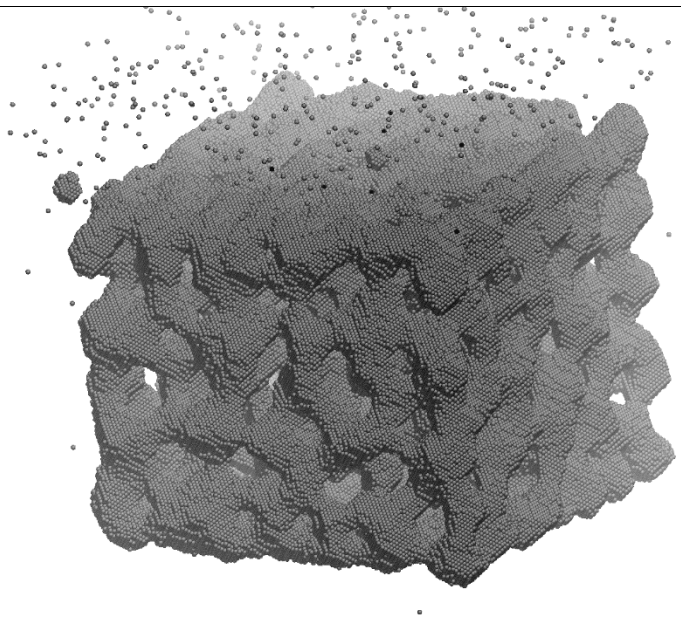


Рис. 7. Результат спікання нанокластерів металу за динаміки α *2,0->(7000 кроків)->0,7->(14000 кроків)->2,0*

α min = 0,8

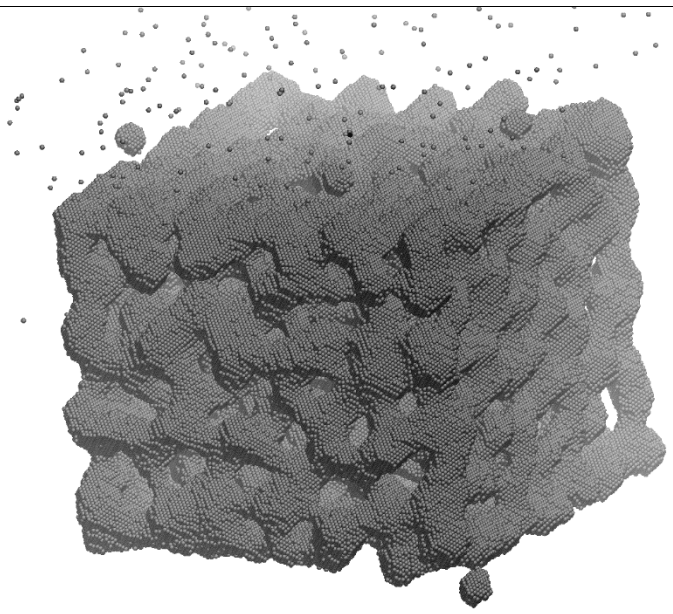


Рис. 8. Результат спікання нанокластерів металу за динаміки α *2,0->(7000 кроків)->0,8->(14000 кроків)->2,0*

α min = 0,9

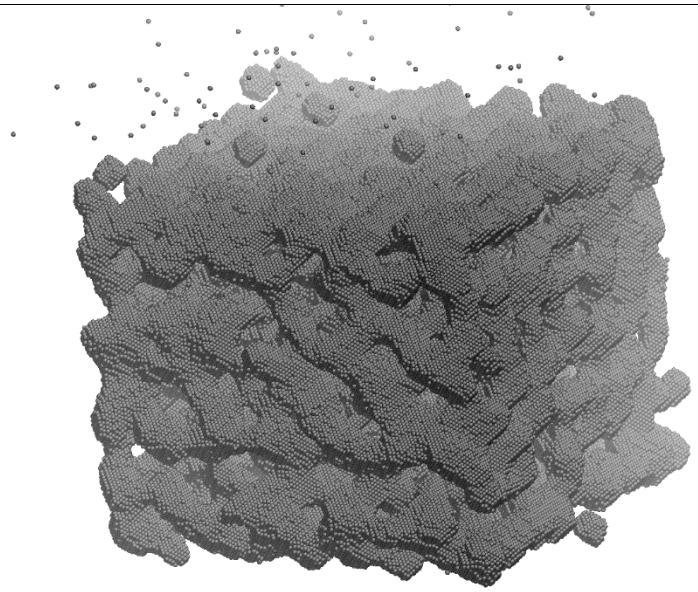


Рис. 9. Результат спікання нанокластерів металу за динаміки α *2,0->(7000 кроків)->0,9->(14000 кроків)->2,0*

α min = 1,0

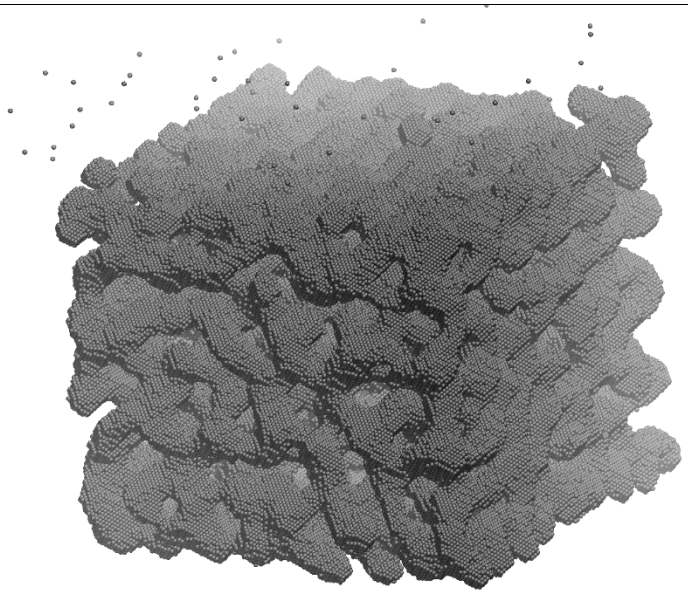


Рис. 10. Результат спікання нанокластерів металу за динаміки α *2,0->(7000 кроків)->1,0->(14000 кроків)->2,0*

α min = 1,1

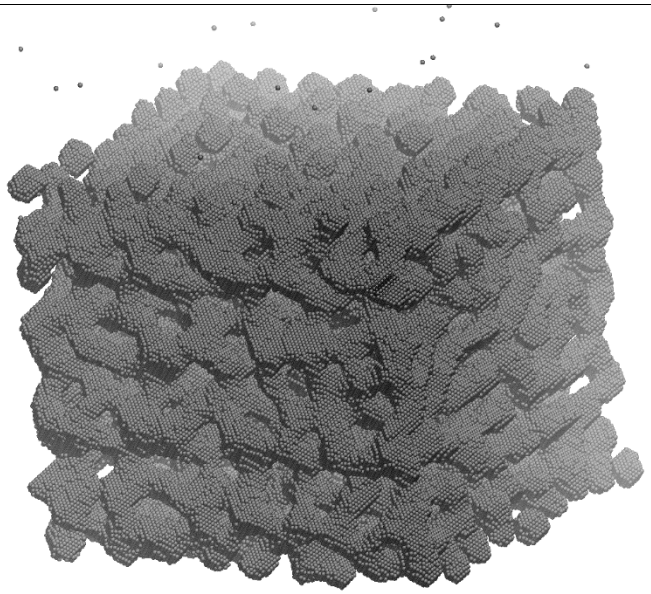


Рис. 11. Результат спікання нанокластерів металу за динаміки α *2,0->(7000 кроків)->1,1->(14000 кроків)->2,0*

α min = 1,2

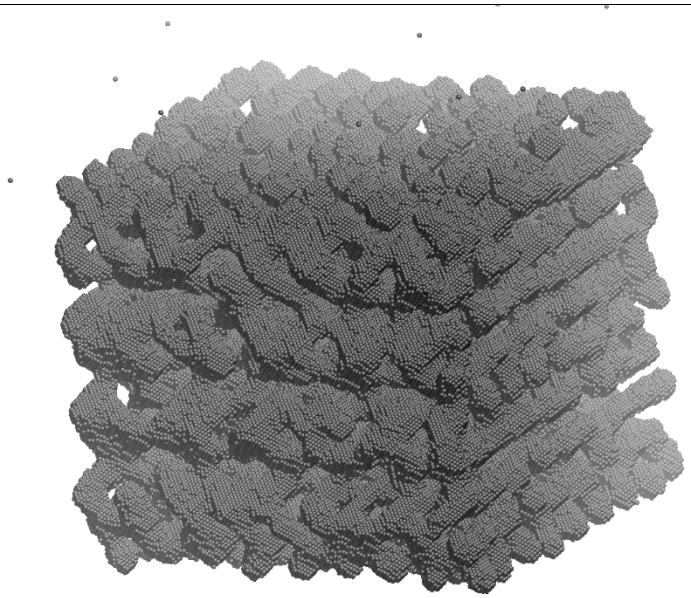


Рис. 12. Результат спікання нанокластерів металу за динаміки α *2,0->(7000 кроків)->1,2->(14000 кроків)->2,0*